

**PROCEEDINGS**  
**of the Second Caltech Conference on**  
**VERY LARGE SCALE INTEGRATION**

held at the  
**California Institute of Technology**  
19—21 January, 1981

Organized by the Caltech Computer Science Department  
and the Caltech Industrial Associates Office  
and sponsored by  
Caltech Industrial Associates  
and the National Science Foundation

Editor: Charles L. Seitz





## TABLE OF CONTENTS

FORWARD	iv
INVITED SPEAKERS SESSION	1
<i>Chairperson: Charles L. Seitz</i>	
The MPC Adventures <i>Lynn A. Conway</i>	5
MOSIS - The ARPA Silicon Broker <i>Danny Cohen, George Lewicki</i>	29
Fast Turnaround Fabrication for Custom VLSI <i>Gunnar A. Wetlesen</i>	45
Longer Term Directions for Semi-Custom VLSI <i>Gordon B. Hoffman</i>	55
FABRICATION SESSIONS	63
<i>Chairperson: James D. Meindl</i>	
Trends in Silicon Processing <i>V. Leo Rideout</i>	65
Electron Beam Testing and Restructuring of Integrated Circuits <i>D. C. Shaver</i>	111
Two Timing Samplers <i>Edward H. Frank, Robert F. Sproull</i>	127
The Role of Test Chips in Coordinating Logic and Circuit Design and Layout Aids for VLSI <i>Martin G. Buehler, Loren W. Linholm</i>	135
INNOVATIVE LSI DESIGNS SESSION	153
<i>Chairperson: Gerald J. Sussman</i>	
Bit Serial Inner Product Processors in VLSI <i>Misha R. Buric, Carver A. Mead</i>	155
A Smart Memory Array Processor for Two Layer Path Finding <i>Christopher R. Carroll</i>	165
Special Purpose Hardware for Design Rule Checking <i>Larry Seiler</i>	197
A VLSI Tactile Sensing Array Computer <i>John E. Tanner, Marc H. Raibert, Raymond Eskenazi</i>	217

## Table of Contents (Cont'd)

COMPUTER-AIDED DESIGN SESSION	
<i>Chairperson: Martin Newell</i>	235
Algorithmic Layout of Gate Macros	
<i>Daniel D. Gajski, Avinoam Bilgory, Joseph Luhukay</i>	237
SLIM: A Language for Microcode Description and Simulation in VLSI	
<i>John Hennessy</i>	253
Signal Delay in RC Tree Networks	
<i>Paul Penfield, Jr., Jorge Rubinstein</i>	269
Functional Verification in an Interactive Symbolic IC Design Environment	
<i>Bryan Ackland, Neil Weste</i>	285
A Methodology for Improved Verification of VLSI Designs Without Loss of Area	
<i>Louis K. Scheffer</i>	299
INNOVATIVE CIRCUIT DESIGNS SESSION	
<i>Chairperson: Thomas F. Knight, Jr.</i>	311
Considerations for an Analog Four Quadrant SC Multiplier	
<i>Phillip E. Allen, William H. Cantrell</i>	313
A One Transistor RAM for MPC Projects	
<i>James J. Cherry, Gerald L. Roylance</i>	329
PLA Design in NAND Structure	
<i>Chong Ming Lin</i>	343
A Multiproject Chip Approach to the Teaching of Analog MOS LSI and VLSI	
<i>Yannis P. Tsividis, Dimitri A. Antoniadis</i>	355
DESIGN DISCIPLINES SESSION	
<i>Chairperson: Martin Rem</i>	373
Towards a Formal Treatment of VLSI Arrays	
<i>Lennart Johnsson, Danny Cohen, Uri Weiser, Alan L. Davis</i>	375
A Notation for Designing Restoring Logic Circuitry in CMOS	
<i>Martin Rem, Carver Mead</i>	399
A Structured Approach to VLSI Layout Design	
<i>M. S. Krishnan</i>	413
Minimum Propagation Delays in VLSI	
<i>Carver Mead, Martin Rem</i>	433

## Table of Contents (Cont'd)

Towards More Realistic Models of Computations for VLSI <i>B. M. Chazelle L. M. Monier</i>	441
A Logic Design Theory for VLSI <i>John P. Hayes</i>	455
ARCHITECTURE SESSION <i>Chairperson: Alan L. Davis</i>	477
A Restructurable Integrated Circuit for Implementing Programmable Digital Systems <i>Rob Budzinski, John Linn, Satish Thatte</i>	481
Communication in a Tree Machine <i>Sally A. Browning, Charles L. Seitz</i>	509
The Torus: An Exercise in Constructing a Processing Surface <i>Alain J. Martin</i>	527
Architecture for VLSI Design of Reed-Solomon Encoders <i>K. Y. Liu</i>	539
Communications for Next Generation Single Chip Computers <i>David R. Smith, Douglas Chan</i>	555

## FOREWORD

As Lynn Conway pointed out in her invited talk (page 5), the two-year period between that first VLSI conference held at Caltech in January 1979 and this Second Caltech Conference on VLSI "...has been one of tremendous activity in VLSI, a time of real discovery and rapid progress."

Let me mention two of the important milestones reached in this period. Regular and reliable channels for those of us in universities to fabricate our designs were established, and several existing and new companies are organizing to provide such services commercially. The building of clean interfaces between design and fabrication, and the possible restructuring and broadening the design base of the microcircuit industry along this pattern, is the theme of the invited speakers session that opened the conference.

*Introduction to VLSI Systems* by Carver Mead and Lynn Conway was published in the Fall of 1980, and, with some stimulus also in the form of "teacher's courses," the VLSI design courses and project laboratories pioneered in a few universities and innovative companies seem since to be spreading exponentially.

That Lynn Conway and Carver Mead were the central figures in both of these accomplishments, and that their energies have been directed at these two projects that have made the VLSI research community more cohesive and cooperative, is surely a testimony to their insight and character.

The technical sessions were organized to provide a broad view -- including fabrication, innovative designs, design tools, design disciplines, and architecture -- of research efforts underway in industry, government, and universities.

The 28 papers presented were selected by the organizing committee from nearly five times as many submitted. We received many more excellent papers than we could accept for presentation and publication.

We at Caltech are very pleased with the alternation established with the MIT Conference on Advanced Research in Integrated Circuits in January 1980, and January 1982, and recommend to the interested reader the proceedings of the January 1982 conference and of other conferences held at the University of Edinburgh (August 1981) and Carnegie-Mellon University (October 1981). Proceedings from these conferences were published by Academic Press (VLSI81), Edinburgh), by Computer Science Press (VLSI Systems and Computations, Carnegie-Mellon), and Artech House, Dedham, MA (Proceedings, Conference on Advanced Research in VLSI, MIT January 1982). Proceedings of the Caltech conferences, January 1979 and January 1981, will continue to be available through the Computer Science Librarian, Caltech 256-80, Pasadena, CA 91125.

Alas, the commercial publishers we approached in the fall of 1980, prior to the publication of the Mead & Conway text, were not yet ready to publish VLSI Proceedings, and we had to undertake this job internally. Your editor greatly regrets and apologizes to the authors and to those who had to wait for their orders to be filled for the extraordinary delays we have experienced in preparing this 600-page document. Very special thanks go to my secretary, Vivian Davies, for her care and persistence in assembling the document, making arrangements with printers, and sorting out the orders, after a turnover in our staff, as well as reminding the editor frequently of his duties.

This conference was organized jointly by the Caltech Computer Science Department and the Caltech Industrial Associates Office, and was sponsored by the Industrial Associates and by the National Science Foundation. My thanks to Bernie Chern of NSF for his support and assistance in expanding the representation at the conference to many more universities.

Finally, let me express my thanks and appreciation to the technical program committee, consisting of Forest Baskett, Xerox PARC and Stanford University; Alan L. Davis, University of Utah; Lee Hollaar, University of Utah; Paul Hudak, University of Utah; Lennart Johnsson, Caltech; Thomas F. Knight, Jr., Massachusetts Institute of Technology; James D. Meindl, Stanford University; Glenn Miranker, IBM Corporation; Martin Newell, Xerox

PARC; Martin Rem, Technical University, Eindhoven, and Caltech; James A. Rowson, Caltech; Dick Sites, Digital Equipment Corporation; Harold Stone, University of Massachusetts; and Gerald J. Sussman, Massachusetts Institute of Technology, for classifying and refereeing the very large number of papers over a brief period before Christmas.

Charles L Seitz

Conference Chairperson and Proceedings Editor

INVITED SPEAKERS SESSION

*Chairperson: Charles L. Seitz  
Associate Professor of Computer Science  
California Institute of Technology*

## INVITED SPEAKERS SESSION

Where the invited speakers session of our 1979 VLSI conference was devoted to a survey of the evolving research areas, for this conference we tried to select a single topic of current interest to people from industry, government, and universities: the "silicon foundry" and "implementation system."

There are many definitions of the "silicon foundry," from simply a factory that fabricates chips "designed elsewhere," to Gordon Hoffman's definition of a semi-custom integrated circuit (page 61) as one that appears custom to the user but standard to the manufacturer. This departure from the situation most often found today, in which designers work for the same company that fabricates their chips, indeed usually in the same building as the fabrication line, is reminiscent of the period in the 1950's when users of computers started in earnest to supplement and specialize the programs provided by computer manufacturers.

The forces behind this possible restructuring of the microcircuit industry are similar: the need to broaden the design capability of the industry, and the differences in business organization that encourage and reward the designer and the foundry (see Carver Mead's article on "VLSI and Technological Innovation" in the 1979 Proceedings).

The technical problems and solutions also may well be similar, and can be described in terms of "clean interfaces" that must be established between design and fabrication. These interfaces may be at many different levels. One expects the inevitable tradeoffs between low-level representations that take longer to design but which achieve very high density and performance, and high-level representations that reduce design time and do not squeeze as much onto a chip as possible. If the fabrication technology is outrunning the designers, reminiscent of course of the situation with computer hardware and software, perhaps we must learn to use this



remarkable fabrication technology in ways that optimize returns rather than silicon.

The five talks in this session -- unfortunately Professor Carver Mead's talk was not captured on tape due to a defective tape cassette -- represent a progression from the rationale for this approach, to the first experimental implementation system, to the first automated production implementation system MOSIS, to a commercial startup to provide fabrication services for custom designs addressing now additional issues in yield, testing, and customer education, and finally to systems in which the interface is elevated from the exchange of mask geometry to function, a two step interface that permits still more independence between design and fabrication.

These papers were reconstructed from tape and/or the author's notes, with an attempt to retain the "first person" spontaneity of the talks, and where mistakes are found, they are certainly the fault of this editor and not of the author. Please let me convey here my appreciation to the authors for sharing their ideas in this conference.



**THE MPC ADVENTURES:  
Experiences with the Generation of  
VLSI Design and Implementation Methodologies**

Lynn A. Conway

Research Fellow, and  
Manager, VLSI System Design Area,  
Palo Alto Research Center, Xerox Corporation

## 1. Introduction

It's great to be here with you today. I remember an equally sunny January day here in Pasadena when the first VLSI Conference was held at Caltech two years ago. That seems such a short while ago, but the period since has been one of tremendous activity in VLSI, a time of real discovery and rapid progress. I'm really looking forward to the Technical Sessions of the next two days, to hearing about some of the best recent work in this exciting field.

My talk today is about "The MPC Adventures", namely the multi-university, MultiProject Chip escapades of the past two years. I'll describe these adventures, and the new VLSI implementation system that made possible the economical, fast-turnaround implementation of VLSI design projects on such a large scale. I'll also describe the experiences I've had with the processes involved in generating new cultural forms such as the "Mead-Conway" VLSI design and implementation methodologies. One of my objectives today is to help you visualize the role that the "MPC Adventures" played in the generation of the methodologies.

I am particularly interested in developing effective research methodologies in the sciences of the artificial, especially in areas such as engineering design. The sort of question that really interests me is: How can we best organize to create, validate, and culturally integrate new design methods in new technologies? What are the research dynamics involved? Consider the following:

When new design methods are introduced in any technology, especially in a new technology, a large-scale exploratory application of the methods by many designers is necessary in order to test and validate the methods. A lot of effort must be expended by a lot of people, struggling to create many different systems, in order to debug the primitives and composition rules of the methodology and their interaction with the underlying technology. A similar effort must also be expended to generate enough design examples to evaluate the architectural possibilities of the design methods and the technology. That is the first point: *A lot of exploratory usage is necessary to debug and evaluate new design methods.* The more explorers that are involved in this process, and the better they are able to communicate, the faster the process runs to any given degree of completion.

Suppose some new design methods have been used and fairly well debugged by a community of exploratory designers, and have proven very useful. Now consider the following question: How can you take methods that are new, methods that are not in common use and therefore perhaps considered *unsound methods*, and turn them into *sound methods*? In other words, how can you

cause the *cultural integration* of the new methods, so that the average designer feels comfortable using the methods, considers such usage to be part of their normal duties, and works hard to correctly use the methods? Such cultural integration requires a major shift in technical viewpoints by many, many individual designers. Changes in design practices usually require changes in the social organization in which the designer functions. These are difficult obstacles to overcome. We see that numbers are important again, leading us to the second point: *A lot of usage is necessary to enable sufficient individual viewpoint shifts and social organization shifts to occur to effect the cultural integration of the methods.* The more designers involved in using the new methods, and the better they are able to communicate with each other, the faster the process of cultural integration runs.

When methods are new and are still considered unsound, it is usually impossible in traditional environments to recruit and organize the large numbers of participants required for rapid, thorough exploration and for cultural integration. Therefore, new design methods normally evolve via rather ad hoc, undirected processes of cultural diffusion through dispersed, loosely connected groups of practitioners, over relatively long periods of time. (Think, for example of the effect of the vacuum-tube-to-transistor technology transition on the design practices of the electronic design community, or of the effect of the discrete-transistor-to-TTL technology transition). When the underlying technology changes in some important way, new design methods exploiting the change compete for market share of designer mind-time, in an ad hoc process of diffusion. Bits and pieces of design lore, design examples, design artifacts, and news of successful market applications, move through the interactions of individual designers, and through the trade and professional journals, conferences, and mass media. When a new design methodology has become widely integrated into practice in industry, we finally see textbooks published and university courses introduced on the subject.

I believe we can discover powerful alternatives to that long, ad hoc, undirected process. Much of this talk concerns the application of methods of experimental computer science to the particular case of the rapid directed creation, validation, and cultural integration of the new VLSI design and VLSI implementation methods within a large computer-communication network community.

First I will sketch the evolution of the new VLSI design methods, the new VLSI design courses, and the role that implementation played in validating the concepts as they evolved. Next I'll bring you up to date on the present status of the methods, the courses, and the implementation systems. Finally, I'll sketch of the methods that were used to direct this evolutionary process. We'll reflect a bit on those methods, and look ahead to other areas where such methods might be applied.

## 2. Evolution of the VLSI Design Courses; Role of the MPC Adventures

In the early 1970's, Carver Mead began offering a pioneering series of courses in integrated circuit design here at Caltech. The students in these courses in MOS circuit design were presented the basics of industrial design practice at the time. Some of these students went on to do actual design projects, and Carver found that even those without backgrounds in device physics were able to complete rather ambitious projects after learning these basics. These experiences suggested that it might be feasible to create new and even simpler methods of integrated system design.

In the mid 1970's, a collaboration was formed between my group at Xerox PARC and a group led by Carver here at Caltech, to search for improved methods for VLSI design. We undertook an effort to create, document, and debug a simple, complete, consistent method for digital system design in nMOS. We hoped to develop and document a method that could be quickly learned and

applied by digital system designers, folks skilled in the problem domain (digital system architecture and design) but having limited backgrounds in the solution domain (circuit design and device physics). We hoped to generate a method that would enable the system designer to really exploit the architectural possibilities of planar silicon technology without giving up the order of magnitude or more in area-time-energy performance sacrificed when using the intermediate representation of logic gates as in, for example, traditional polycell or gate-array techniques.

Our collaborative research on design methodology yielded important basic results during '76 and '77. We formulated some very simple rules for composing FET switches to do logic and make registers, so that system designers could easily visualize the mapping of synchronous digital systems into nMOS. We formulated a simple set of concepts for estimating system performance. We created a number of design examples that applied and illustrated the methods.

### ***The Mead-Conway Text***

Now, what could we do with this knowledge? Write papers? Just design chips? I was very aware of the difficulty of bringing forth a new *system of knowledge* by just publishing bits and pieces of it in among traditional work.

I suggested the idea of writing a book, actually of *evolving a book*, in order to generate and integrate the methods, and in August 1977 Carver and I began work on the Mead-Conway text. We hoped to document a complete, but simple, system of design knowledge in the text, along with detailed design examples. We quickly wrote a preliminary draft of the first three chapters of this text, making use of the Alto personal computers, the network, and the electronic printing systems at PARC. In parallel with this, Carver stimulated work on an important design example here at Caltech, the work on the "OM2". Dave Johannsen carefully applied the new design methods as they were being documented, refined and simplified, to the creation of this major design example.

We then decided to experimentally debug the first three chapters of material by interjecting them into some university MOS design courses. An initial draft of the first three chapters<sup>1(a)</sup> was used by Carlo Sequin at U.C. Berkeley, and by Carver Mead at Caltech in the fall of '77. During the fall and winter of '77-'78, Dave Johannsen finished and documented the new OM2 design. The OM2 provided very detailed design examples that were incorporated into a draft of the first five chapters<sup>1(b)</sup> of the text. We distributed that draft in February '78 into spring semester courses by Bob Sproull at CMU, and by Fred Rosenberger at Washington University, St. Louis.

We were able to debug and improve the material in these early drafts by getting immediate feedback from the '77-'78 courses. We depended heavily on use of the ARPAnet for electronic message communications. Our work rapidly gained momentum. A number of people joined to collaborate with us during the spring of '78: Bob Sproull at CMU and Dick Lyon at PARC created the CIF 2.0 specification; Chuck Seitz prepared the draft of Chapter 7 on self-timed systems; H. T. Kung and several others contributed important material for Chapter 8 on Concurrent Processing. By the summer of '78 we completed a draft of the manuscript of the entire textbook.<sup>1(c)</sup>

### ***The MIT'78 VLSI Design Course***

During the summer of 1978, I prepared to visit M.I.T. to introduce the first VLSI system design course there. This was to be a major test of the full set of new methods and of a new intensive, project-oriented form of course. I also hoped to thoroughly debug the text prior to publication. I wondered: How could I really test the methods and test the course contents? The answer was to

spend only half of the course on lectures on design methods; then in the second half, have the students do design projects. I'd then try to rapidly implement the projects and see if any of them worked (and if not, find out what the bugs were). That way I could discover bugs, or missing knowledge, or missing constraints in the design methods or in the course curriculum.

I prepared a detailed outline for such a course, and printed up a bunch of the drafts of the text. Bob Hon and Carlo Sequin organized the preparation of a "Guide to LSI Implementation"<sup>2</sup> that contained lots of practical information related to doing projects, including a simple library of cells for I/O pads, PLA's, etc. I then travelled to M.I.T., and began the course. It was a very exciting experience, and went very well. We spent seven weeks on design lectures, and then an intensive seven weeks on the projects. Shortly into the project phase it became clear that things were working out very well, and that some amazing projects would result from the course.

While the students were finishing their design projects, I cast about for a way to get them implemented. I wanted to actually get chips made so we could see if the projects worked as intended. But more than that, I wanted to see if the whole course and the whole method worked, and if so, to have demonstrable evidence that it had. So I wanted to take the completed layout descriptions and very quickly turn them into chips, i.e. implement the designs (We use the term "VLSI implementation" for the overall process of merging the designs into a starting frame, converting the data into patterning format, making masks, processing wafers, dicing the wafers into chips, and mounting and wire-bonding the chips into packages).

We were fortunate to be able to make arrangements for fast implementation of those student projects following the MIT course. I transmitted the design files over the ARPAnet from M.I.T. on the east coast to some folks in my group at PARC on the west coast. The layouts of all the student projects were merged together into one giant multiproject chip layout, a trick developed here at Caltech, so as to share the overhead of maskmaking and wafer fab over all of the designs. The project set was then hustled rapidly through the prearranged mask and fab services. Maskmaking was done by Micro-Mask, Inc., using their new electron-beam maskmaking system, and wafer fabrication was done by Pat Castro's Integrated Circuit Processing Lab (ICPL) at HP Research, in Palo Alto. We were able to get the chips back to the students about six weeks after the end of the course. A number of the M.I.T. '78 projects worked, and we were able to uncover what had gone wrong in the design of several of those that didn't.

The M.I.T. course led to a very exciting group of projects, some of which have been described in later publications. I'll now show a map and some photos of the project chip (see Ref. 6). The project by Jim Cherry, a transformational memory system for mirroring and rotating bit map image data, is particularly interesting, and was one of those that worked completely correctly. Jim's project is described in detail in the second edition of the Hon and Sequin Guidebook (see Ref. 5). Another interesting project is the prototype LISP microprocessor designed by Guy Steele, that was later described in an M.I.T. AI Lab report.<sup>3</sup>

As a result of this course and the project experiences, we uncovered a few more bugs in the design methods, found constraints that were not specified, topics that were not mentioned in the text, that sort of thing. You can see that the project implementation did far more than test student projects. It also tested the design methods, the text, and the course.

During the spring of '79 we began preparing the final manuscript of the Mead-Conway text for publication by Addison-Wesley the following fall.<sup>4</sup> Hon and Sequin began preparing a major



revision of the Implementation Guide<sup>5</sup> that would contain important things like a CIF primer, new, improved library cells, and so forth. I began preparing an "Instructor's Guide", based on the experiences and information from the M.I.T. '78 VLSI design course,<sup>6</sup> containing a detailed course outline, a complete set of lecture notes, and homework assignments from that course. These materials would help transport the course to other environments.

### *The MPC Adventures: MPC79 and MPC580*

I'll now describe the events surrounding the multiproject chip network adventures of the fall of 1979 and spring of 1980. I remember thinking: "Well, ok, we've developed a text, and also a course curriculum that seems transportable. The question now is, can the course be transported to many new environments? Can it be transported without one of the principals running the course?" In reflecting on the early work on the text by communicating with our collaborators via the ARPAnet, and by thinking about which schools might be interested in offering courses, I got an idea: If we could find ways of starting project-oriented courses at several additional schools, and if we could also provide VLSI implementation for all the resulting student projects, we could conduct a really large test of our methods. The course might be successful in some schools, and not in others, and we could certainly learn a lot from those experiences. I began to ponder the many ways we could use the network to conduct such an adventure.

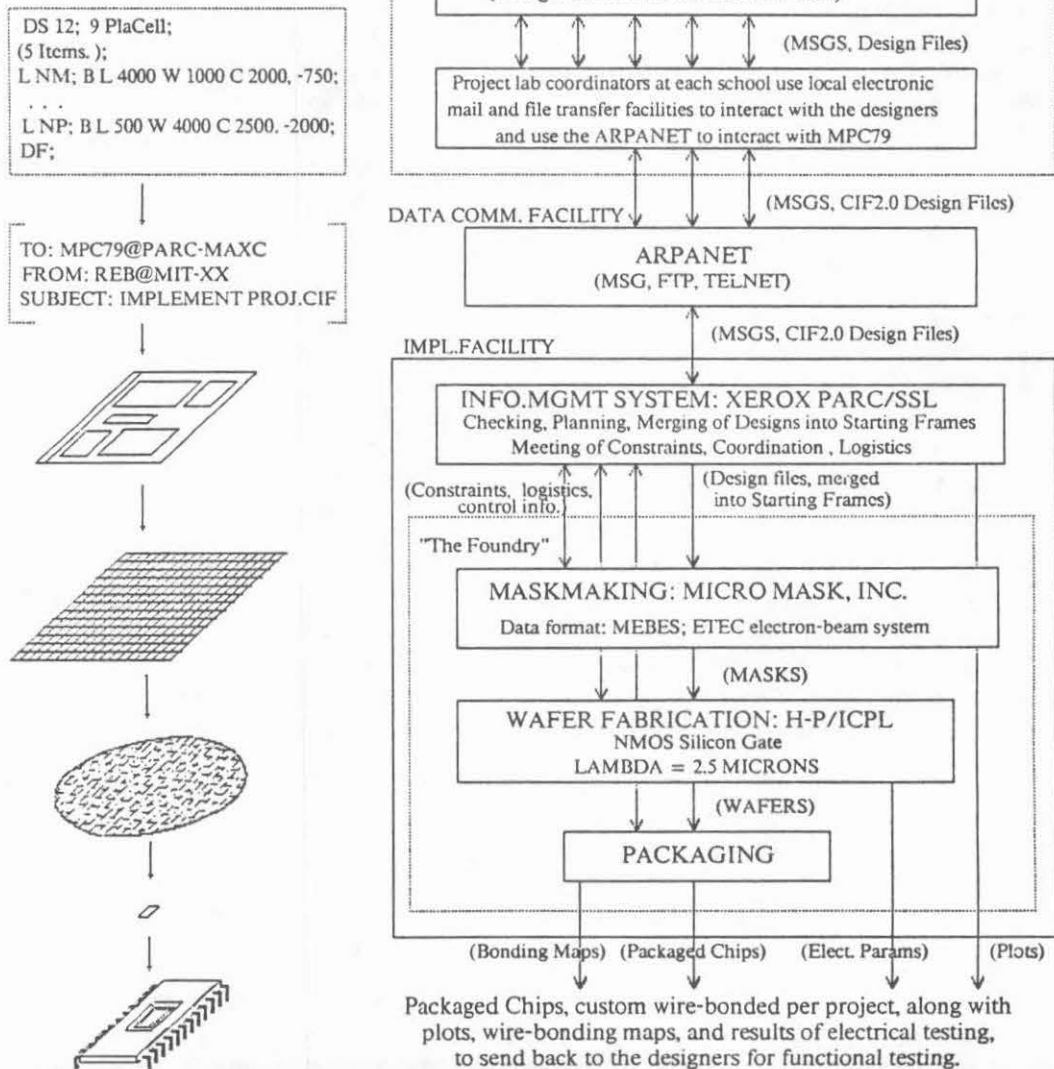
We began to train instructors from a number of universities in the methods of teaching VLSI design. Doug Fairbairn and Dick Lyon ran an intensive short course for PARC researchers during the spring of '79, and a videotape<sup>7</sup> was made of that entire course. During the summer of '79, we began using those tapes as the basis for short, intensive "instructor's courses" at PARC for university faculty members. Carver Mead and Ted Kehl also ran an instructor's course at the University of Washington, with the help of the PARC tapes, in the summer of '79. All "graduates" of the courses received copies of the Instructor's Guide, to use as a script at their schools.

By early fall of '79, quite a few instructors were ready to offer courses. We at PARC gathered up our nerve, and then announced to this group of universities: "If you run courses, we will figure out some way so that at the end of your course, on a specified date, we will take in any designs that you transmit to us over the ARPAnet; we will implement those projects, and send back wire-bonded, packaged chips for all of your projects within a month of the end of your course!" This multi-university, multiproject chip implementation effort came to be known as "MPC79".

About a dozen universities joined to participate in MPC79. As this large university community became involved, the project took on the characteristics of a great "network adventure", with many people simultaneously doing large projects to test out new ideas. Through the implementation effort, students hoped to validate their design projects, instructors would be able to validate their offering of the course, and we would be able to further validate and test the design methodology and the new implementation methods in development at PARC.

We coordinated the MPC79 events by broadcasting a series of detailed "informational messages" out over the network to the project lab coordinators at each school. MSG #1 announced the service and the schedule; MSG #2 distributed the basic library cells, including I/O pads and PLA cells; MSG #3 described the "User's Guide" for interactions with the system; MSG #4 contained information about the use of CIF2.0; MSG #5 provided last-minute information just prior to the design deadline; MSG #6 was sent just after the implementation was completed, and contained news about the results of the entire effort. Figure 1 flowcharts the overall activity.

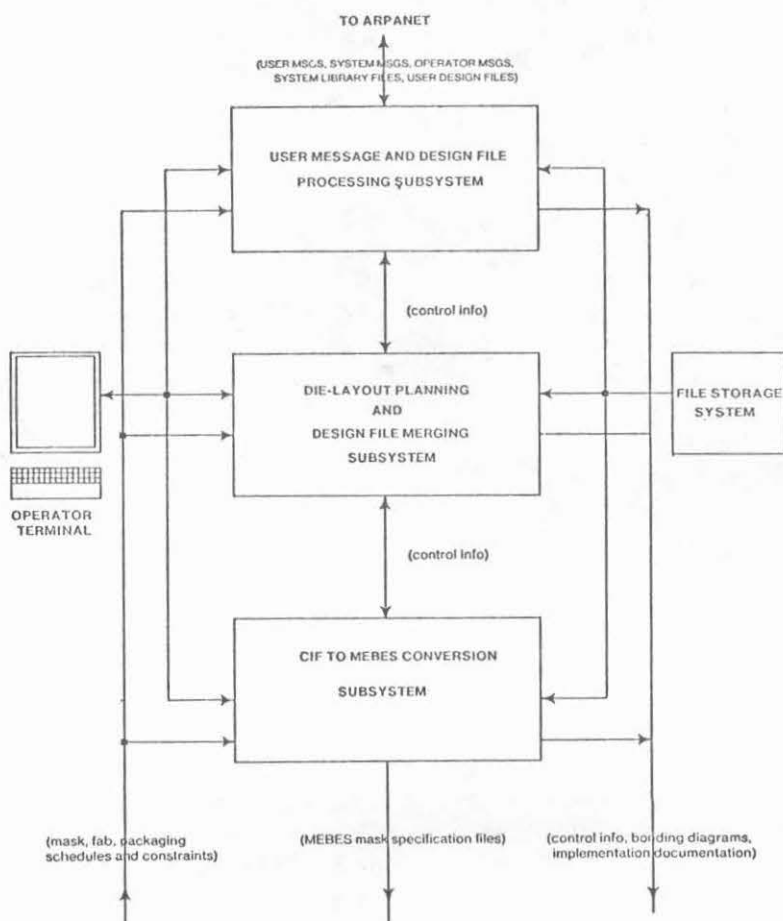
FIGURE 1.

**MPC79 Flowchart:**



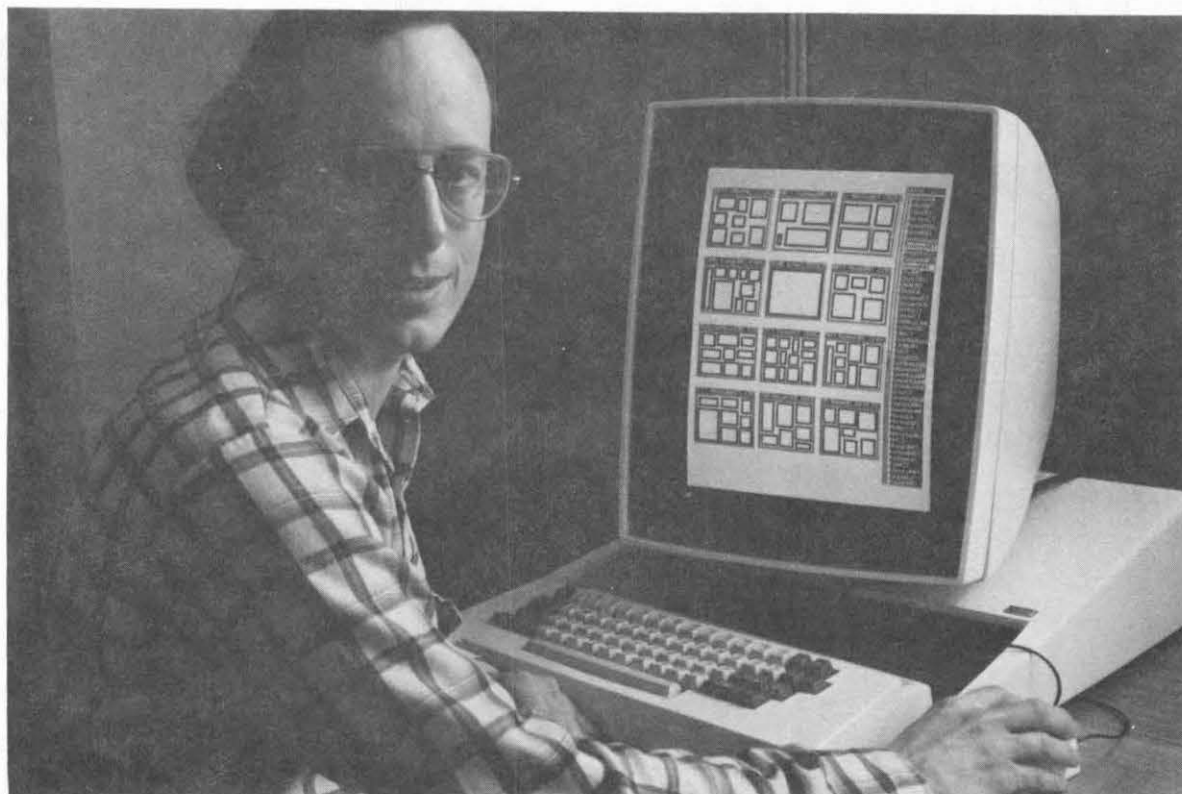
During this period, Alan Bell pioneered the architecture and teamed up with Martin Newell to develop a "VLSI Implementation System", which is something like a time-sharing operating system, or information management system, for providing remote access to mask and fab services. This system manages all user interactions, manages the data base of design files, handles the logistics, the scheduling, enabling users all around the country to interact by electronic messages with (what they perceive to be) an automatic system that implements their projects.

Figure 2 shows a simple block diagram of the basic modules of the system. It contains a user message handler and an associated design file processing subsystem; these provide a means for interacting with users to receive requests for service, transmit status and error messages, and build the design-file data base. It also contains a die-layout planning and design-file merging subsystem used to pack all of the participants designs together into a mask specification following the design deadline time. Finally it contains a CIF to MEBES (electron beam maskmaking) format-conversion subsystem to prepare the data files for hand off to the foundry.



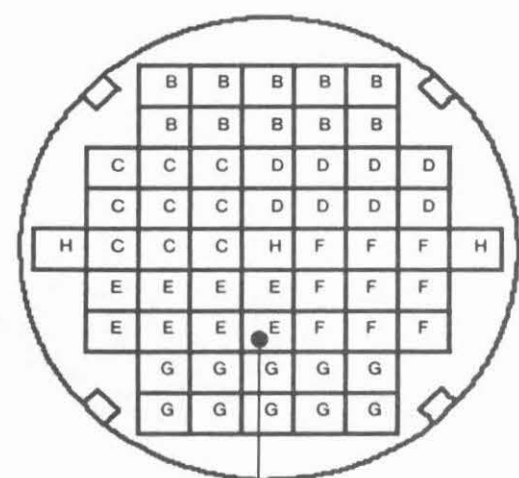
**Figure 2. Block Diagram of the VLSI Implementation System**

Following is a photo (Fig. 3) of Alan Bell operating the implementation system at PARC during the very final stages of project merging following the MPC79 design deadline. He's taken almost all of the designs, as identified in a display menu listing the project ID's, and packed them into the 12 die-types of the project set.

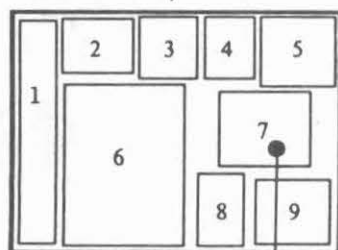


**Figure 3. Alan Bell using the Implementation System to merge the MPC79 projects**

For MPC79, the implementation system produced MEBES mask specifications containing 82 projects from 124 participating designers, merged into 12 die-types that were distributed over two mask sets. Thus there was a tremendous sharing of the overhead involved in the maskmaking and wafer fab. For MPC79 the masks were again made by Micro-Mask, Inc., and wafer fabrication was again done by HP-ICPL. Several chips of each project type were custom wire-bonded and prepared for shipment back to the designers, along with "implementation documentation"<sup>8</sup> containing pinout information for the projects, electrical parameter measurements for the wafer lots, etc. Figure 4 provides a visualization of the many projects conveyed through one of the MPC79 wafer types, and of the corresponding hierarchy of information associated with the project set.



AE



MPC79 AE

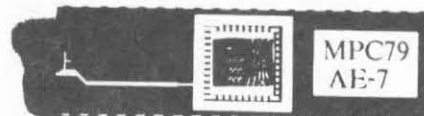
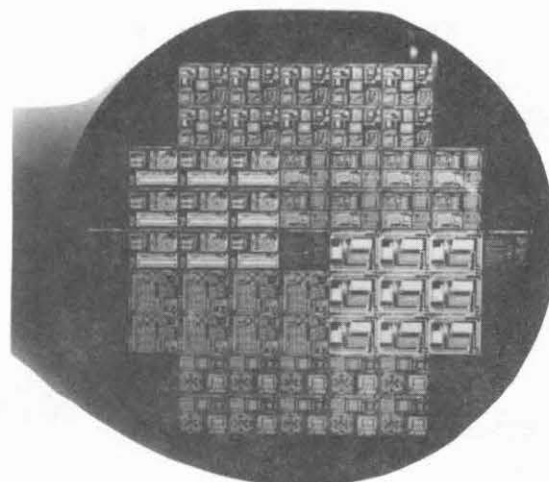
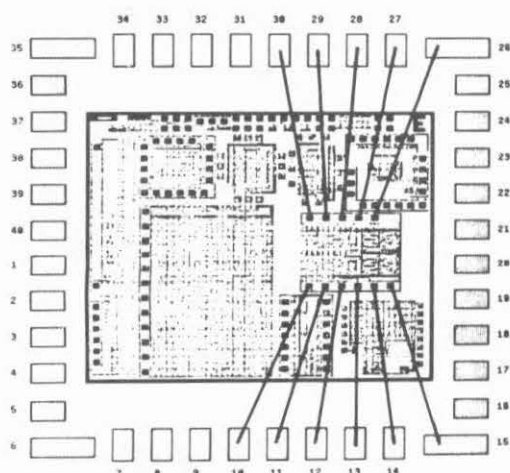
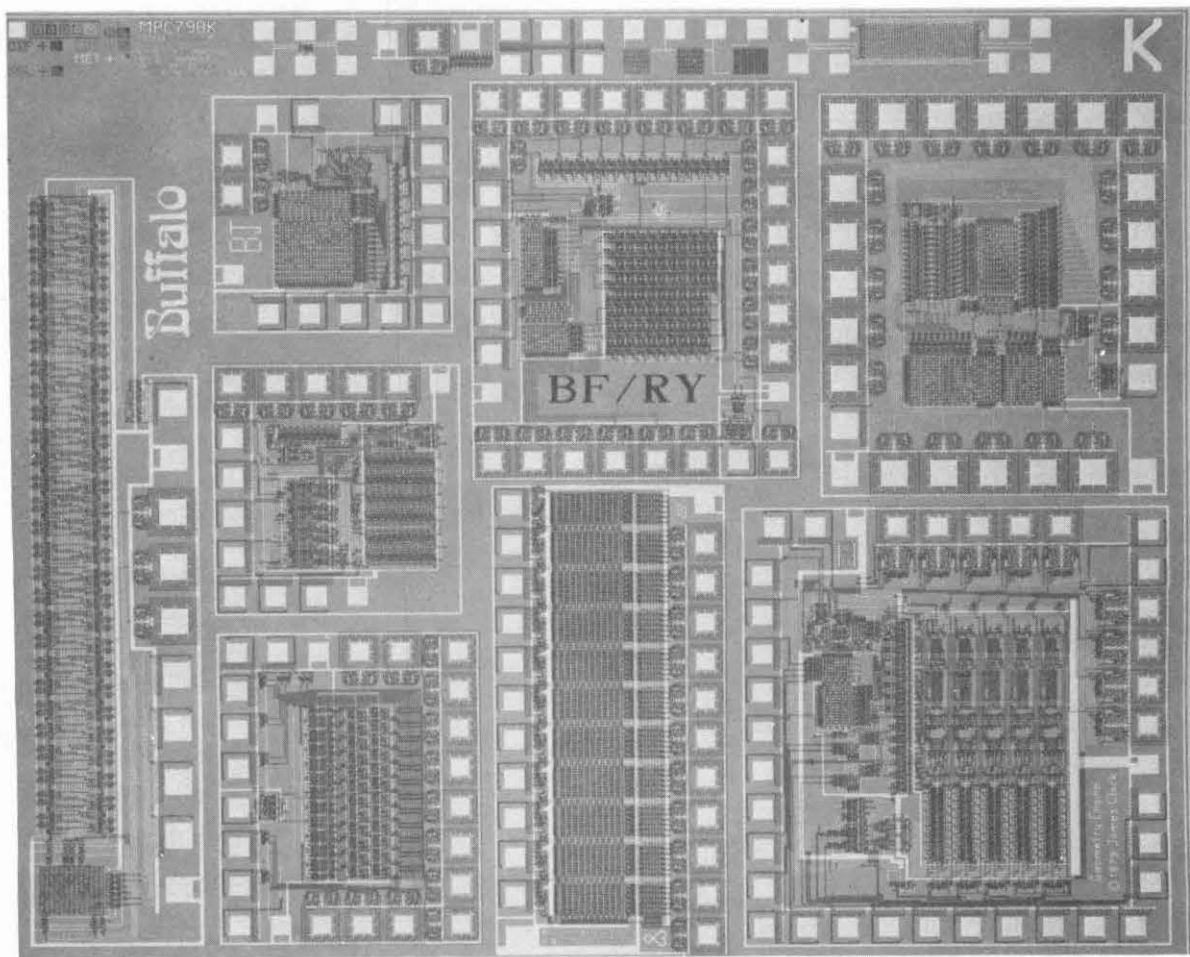


Figure 4.

Above: Photo of MPC79 type-A wafer, type-AE die, type AE-7 chip.

At Left: Corresponding hierarchy of informational material.

Just 29 days after the design deadline time at the end of the courses, packaged custom wire-bonded chips were shipped back to all the MPC79 designers. Many of these worked as planned, and the overall activity was a great success. I'll now project photos of several interesting MPC79 projects. First is one of the multiproject chips produced by students and faculty researchers at Stanford University (Fig. 5). Among these is the first prototype of the "Geometry Engine", a high-performance computer graphics image-generation system, designed by Jim Clark. That project has since evolved into a very interesting architectural exploration and development project.<sup>9</sup>



**Figure 5. Photo of MPC79 Die-Type BK (containing projects from Stanford University)**

Another project that turned up in MPC79 was a LISP microprocessor<sup>10</sup> designed by Holloway, Sussman, and Steele at MIT and Bell at PARC. This "Scheme-79" chip is a further step in the evolution of LISP microprocessor architectures by the M.I.T. AI-Lab group. Their work is based on the prototype LISP microprocessor<sup>3</sup> Guy Steele designed for the 1978 MIT course.

The results of this design methodology experimentation and demonstration were very exciting, and convinced us of the overall merits of the design methods, the courses, and the implementation infrastructure. We first reported on the results at the M.I.T. VLSI conference in January 1980.<sup>11,12</sup>

At PARC we then began the transfer of the implementation system technology to an internal operational group; the transfer was completed during the spring of 1980. That operational group now has the responsibility of providing VLSI implementation service within Xerox. They ran the implementation system for a very large group of schools in the spring of 1980, in order to provide themselves with a full-scale test the overall operation of the system, and to confirm the success of the technology transfer. That effort, known as "MPC580"<sup>13</sup>, had about twice as many participants as did MPC79. Over 250 designers were involved! They produced so many projects, including a number of full-die sized projects, that 5 mask sets were required. Although MPC580 involved a lot of maskmaking and wafer fabrication, the project set was turned around from design-cutoff to packaged chips in about six weeks.

Some really interesting projects were created by the MPC580 designers. An example is the RSA encryption chip<sup>14</sup> designed by Ron Rivest at MIT. Ron is a computer science theoretician and faculty member at M.I.T., had taken the VLSI design course the previous fall, and had done a small project for MPC79. He and several other M.I.T. people then created the prototype RSA encryption chip architecture and design during the spring of 1980, in time for the MPC580 cutoff.

I think you can now begin to see the role the provision of implementation plays in stimulating architectural exploration, the offering of design courses, and the creation of design environments.

### **3. Present Status of the VLSI Design Courses and the VLSI Implementation Systems**

The design methodology introduced in the Mead-Conway text has now become well integrated into the university computer science culture and educational curriculum. During the '79-'80 school year, courses were offered at about 12 universities. During the present '80-'81 school year, courses are being offered at more than 80 universities.

In addition, a number of industrial firms have begun to offer internal, intensive courses on the design methodology. For example, courses are being offered at several locations within Hewlett-Packard, under the leadership of Merrill Brooksby, Manager of Corporate Design Aids at HP. The HP courses are project oriented, and provide students with fast-turnaround project implementation. Brooksby believes that in addition to directly improving the skills of HP designers, the course plays an important role by providing a common internal base of design knowledge through which designers can communicate about work in other technologies (the "common culture effect"). Similar courses are being offered at DEC, in an effort led by Lee Williams. Many other industrial firms have begun using an excellent videotape VLSI system design course produced recently by VLSI Technology, Inc. (VTI).<sup>15</sup>

Design aid concepts and software are evolving rapidly in the university VLSI research community. During the work on MPC79, we began to see very interesting new types of analysis aids originating at MIT. I'm thinking of the work of Clark Baker, Chris Terman, and Randy Bryant who began creating circuit extractors, static checkers, and switch simulators of a sort appropriate for our design methods.<sup>16,17</sup> They began to provide access to such analysis aids over the network, aids that could be easily and efficiently used to partially validate projects prior to implementation. These tools were used to debug some large projects prior to submission to MPC79 (for example, the Scheme-79



chip). Some of these tools are now in routine use at a number of other universities. I believe we'll soon see analysis aids embodying these new concepts placed into widespread use in industry.

A VLSI implementation system has been put into use by Xerox Corporate Research to support exploratory VLSI system architecture and design within Xerox Corporation. Another implementation system is being operated by USC/ISI for the Defense Advance Research Projects Agency's (DARPA) VLSI research community, a community consisting of several large research universities (including M.I.T., CMU, Stanford, U.C. Berkeley, Caltech, etc.), and a number of Defense Department research contractors. (Danny Cohen will describe that system in a later talk)

The initial system architecture of the system used for MPC79, and the operational experiences during MPC79, provided the knowledge on which the new Xerox and ISI systems were based. One of the major improvements contained in both these newer systems is the fully-automated handling of user electronic message interactions and management of the design file data base. During MPC79, Alan Bell interacted with the designers with some machine assistance in message handling (using a menu-based graphical interface that made message-processing and file management interactions easy and fast), but in fact he did actually look at all user messages. When we ran MPC79, we couldn't predict the bounds on the information that would have to be conveyed between designer and system. The generation of that knowledge was an important result of MPC79, making it possible to automate the message handling and data base management in later systems. Our knowledge about the implementation system to foundry interface was also considerably expanded and refined during these experiences.<sup>18</sup>

As I think back over the origins of the VLSI implementation system, it's clear that we didn't initially set out to create such a system. It was really a serendipitous result. We were extremely motivated and driven to provide VLSI implementation to a large university community. I thought that it just might be possible to do that. I realized that pulling off VLSI implementation on such a vast scale would generate and propagate a lot of artifacts, and thus announce the presence of the new design culture, and help to culturally integrate our methods. So, we began working very hard at PARC to create ideas to bring down the cost per project and the overall turnaround time, and to scale up capabilities for handling as many designers as possible.

Somewhere along the line I began to use the metaphor that "we're creating something for mask and fab that was like the time-shared operating system was for computing systems". Our idea was to create a system that provided remote-entry, time and cost-sharing access to expensive capital equipment, and that also managed the logistics of providing such access to a large user community.

At that time, and even now in most integrated circuit design environments, the maskmaking and wafer fabrication required to implement prototypes for a design project cost about \$15,000 to \$20,000, and with some luck take only three or four months getting through the various queues. (Designers using internal company facilities may not see those costs, but I guarantee they're there; on the other hand, all IC designers are familiar with those long turnaround times). With that as background, we were really amazed when we added up the costs in dollars and time to implement the projects in MPC79. By using the implementation system to provide shared access for a large community of users to what amounts to a "fast-turnaround silicon foundry" for rapid maskmaking and wafer fabrication, we achieved a cost per project on the order of a few hundred dollars, and a total turnaround time of only 29 days! (And remember, we weren't using internal mask and fab facilities at PARC, but were instead going to outside foundry services.)

TABLE 1.  
Computing and Design Environments for 1980-81 VLSI Design  
Courses at Universities that participated in MPC79/MPC580.  
[Reprinted with permission of LAMDA, The Magazine of VLSI Design<sup>19</sup>]

UNIVERSITY:	MIT	Caltech	Stanford	CMU	U.C.B.	U. of Cal. (C.S.)	U. of Illinois	U. of Wash.	U. of Rochester	UCLA	Wash. U. (S.L.)	U.S.C.
<b>COURSE INFORMATION</b>												
Instructor(s)	J. Allen, L. Gladser	C. Mead, C. Sletiz	J. Newkirk, R. Mathews	R. Sproull	R. Newton C. Sequin	J. Murray	J. Abraham E. Davidson	T. Kehl	E. Kinnen G. Kedem	V. Tyree	F. Rosenberger	J. Nelson
Course #	6.371	CS161, CS182	EE271	15-846	CS248	EE594	EE325	CS960D	492	M258A, B, C	EE463	EE599
Sem. or Qtr.	F, Sp	F-W-Sp	F, Sp	Sp	F	F	F, Sp	F, W, Sp	F	F-W-Sp	Sp	F
#stud./Class	35	40	60	30	50	20	20	15	25	20	25	30
<b>COMPUTING ENVIRONMENT</b>												
CPU	DEC-20	DEC-20, VAX	DEC-VAX	DEC-VAX	DEC-VAX	DEC-20	HP1000	DEC-20, VAX	ALTO, VAX	DEC-VAX	DEC-20	DEC-KL10
Op. Sys.	TOPS-20	TOPS-20, UNIX	UNIX	UNIX	UNIX	TOPS-20	RTE IV	TOPS-20, VMS	ALTO, UNIX	UNIX	TOPS-20	TOPS-10
Prog. Lang.	LISP, APL, CLU	Simula, C	C	C	C	Simula, Pascal	Pascal	FORTRAN	C, Pascal	C, Pascal	Simula, FORTRAN	Pascal
<b>DESIGN AID ENVIRONMENT</b>												
Synthesis aids	PLAG, MI	MG, MI	PLAG	PLAG	PLAG, SGC	MG, MI	---	PLAG, MI	---	---	PLAG	---
Description	SLL	SLL	SLL	SLL, IGL	SLL, IGS	SLL	IGL	SLL	IGL	SLL	SLL	SLL
Analysis aids	CX, SS, DRC, CS	CX, SS, CS	CX, DRC, SS	CX, SS, DRC	CX, SS, CS	CS	CS	DRC	CX, SS, DRC	CS	CS	CS
Viewing aids	CPP, BRP	CPP, CRP, CD	BRP	CPP, BRP, CD, BD	BRP, BD	CPP	CPP, BD	CPP	CPP, BRP, CD, BD	CPP	CPP, CD	CPP
Testing aids	MTE	MTE	MTE	---	---	---	MTE	---	MTE	MTE	---	---
<b>PROJECT EXPERIENCE</b>												
(# Projects, # designers)												
MPC79	15, 27	24, 28	19, 35	5, 5	4, 4	1, 1	5, 8	1, 3	5, 9	---	---	---
MPC580	11, 13	21, 22	32, 59	12, 17	8, 12	12, 21	8, 13	15, 15	3, 3	9, 9	9, 11	10, 15

**SUMMARY OF DESIGN-AID CODES:**  
BD: B/W Display; BRP: B/W Raster Plotter; CD: Color Display; CPP: Color Pen Plotter; CRP: Color Raster Plotter; CS: Circuit Simulator; CX: Circuit eXtractor; DRC: layout Design Rule Checker; IGL: Interactive Graphic Layout; IGS: Interactive Graphic Sinks; MG: Module Generator; MI: Module Interconnector; MTE: Minimal Test Environment; PLAG: PLA generator; SGC: Sinks-to-layout Generator/Compressor; SLL: Symbolic Layout Language; SS: Switch Simulators; SSI: Symbolic Sinks Language

Thus we had demonstrated that the time and cost to implement a prototype VLSI designs were as low as they would be using TTL for an equivalent size designs. However, once you've successfully prototyped a design in VLSI, you can take tremendous advantage of the low replication costs and high-performance of VLSI when competing against similar systems implemented in TTL. Therefore, I believe that in addition to the many business opportunities in VLSI design aids and chip designs, there must also be a substantial business opportunities in the area of VLSI implementation systems and services, foundry service brokerage, and foundry services.

Those of you who are interested in learning more about the present courses and design aid environments in the universities might read my recent column<sup>19</sup> in Lambda Magazine. I'll now show a table (see Table 1.) from that article that tabulates the courses, the computing and design-aid environments (as of summer 1980), and the project experience at the key group of 12 universities that collaborated with us at PARC during MPC79 and MPC580. You can see some interesting patterns of diffusion and convergence in this table. You can see how new types of analysis aids are being used this year at most schools to qualify projects for implementation, and how rapidly those new concepts have swept through this university community, most of whom are on the ARPAnet.

#### 4. Sketch of and Reflections on the Research Methods Used

How was all of this done? Let's reflect on these events, focussing on the research methods used to direct and help all of these different things jointly evolve. You'll notice a common idea running through all of these events: Fast-turnaround implementation provides a means for testing concepts and systems at many levels. It isn't just used for testing the project chips. It also tests the design environments, the courses and instructional methods, the text materials, and the design methods.

I'll now describe a basic method of experimental computer science, and sketch how this method was applied to the generation of the VLSI design and implementation methodologies. Later I'll describe the resources required in order to direct this sort of large scale, experimental evolution of engineering knowledge and design practices.

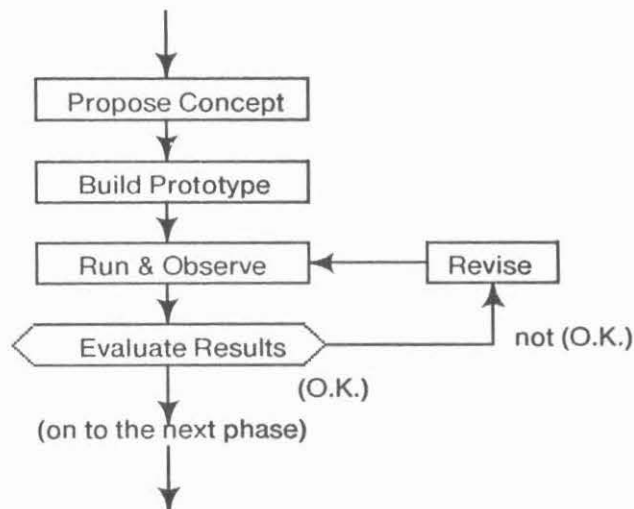
##### *Experimental Method*

There is a basic experimental method that is used in experimental computer science when we are exploring the space of what it is possible to create. The method is especially applicable when creating computer languages, operating systems, and various kinds of computing environments, i.e., applications where we provide primitives that many other people will use to generate larger constructs. Suppose that you've conceived of a new system concept, and want to try it out experimentally. The method is simple: You build a prototype of a system embodying that concept, run the system, and observe it in operation. You might immediately decide, "Hey, this is just not feasible," and scrap the idea right there; or you may think, "Well, maybe we can improve things," or, "Let's try something slightly different," make some revisions, and run the system again. This simple, iterative procedure is sketched in Figure 6. After the experimentation has generated sufficient knowledge (for example, has demonstrated the feasibility of the concept), you may make a transition into some later phase in the evolution of the concept.

What might such later phases be? Suppose you've successfully taken a new concept through a feasibility test, perhaps experimenting with a quick implementation that you ran yourself. You may think, "Well, let's build an improved prototype, and have some other user run it. I'll watch the user use it, and see what happens." After going around that loop a few times, and making further

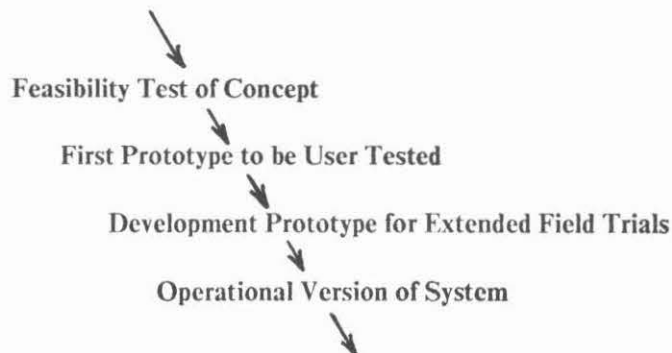


refinements, you may make the transition to building a prototype to be placed into extensive field trials by many users. Thinking back, you can see how the design course was taken through a



**Figure 6. An Experimental Method**

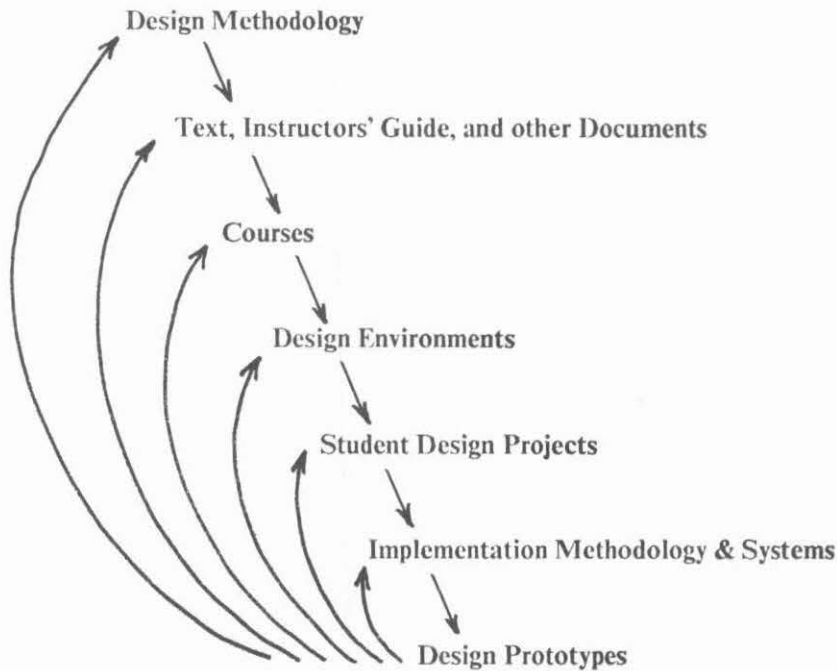
succession of such phases, from feasibility to transfer to a few other "users" and on to full scale "field trials". By obtaining feedback from users and observing results at each step, you move on to on the next phase (see Fig. 7) of refinement and integration of that particular system.



**Figure 7. Some Phases in the Evolution of a System**

If we study the development of the VLSI design methodology, its validation, and its social propagation, you'll notice that the following has happened: The evolution of the methodology involved a multilevel cluster of systems that were being jointly evolved (see Fig. 8). Each system in the cluster runs through the experimental loops, and passes through the various phases of its own evolution. Entries at the higher levels, for example the methodology, or the text, or the documents to support a course, might be more solid and in later phases of their evolution at any given time than, for example, a course in a particular school, or the design environment for that course.

Student design projects play a key role in this process, supporting new refinements in the higher level systems in the hierarchy every new school semester. Fast turnaround implementation of designs was used to close the experimental loop on all the systems in this hierarchy.



**Figure 8. The Joint Evolution of the Multi-Level Cluster of Systems**

If we think back over the evolution of these systems, we can see how all these things were running in parallel in a rapidly enlarging social enterprise. The early courses run here at Caltech demonstrated that it might be feasible to create a simple design methodology. Following the period of basic design methodology research, the preliminary courses run at Caltech, U.C. Berkeley, and CMU helped debug the emerging text documenting the new design methods. The newly documented methodology was then introduced into the M.I.T. '78 course, which became the prototype for the new type of intensive, project-oriented courses. The results of that course prepared the way for seeding similar courses in many other schools.

The text itself passed through drafts, became a manuscript, went on to become a published text. Design environments evolved from primitive CIF editors and CIF plotting software on to include all sorts of advanced symbolic layout generators and analysis aids. Some new architectural paradigms have begun to similarly evolve. An example is the series of designs produced by the OM project here at Caltech. At MIT there has been the work on evolving the LISP microprocessors.<sup>3,10</sup> At Stanford, Jim Clark's prototype geometry engine, done as a project for MPC79, has gone on to become the basis of a very powerful graphics processing system architecture,<sup>9</sup> involving a later iteration of his prototype plus new work by Mark Hannah on an image memory processor.<sup>20</sup>

While these things were evolving, Dick Lyon undertook the important work of developing, debugging, and evolving a set of basic library cells (see refs. 2,5) that would later be used in all of the courses by all of the students in the MPC adventures. Again, in parallel with that, there was the iterative evolution through a series of experiments, from the early multiproject chip sets to the remote entry multiproject chip done at MIT, to the early implementation systems at PARC, and now on to the automated implementation systems at PARC and USC-ISI.

One thing to remember about this is that such enterprises are organized at the meta-level of research methodology and social organization; they are not planned in fully-instantiated detail using some sort of PERT chart. The evolution of a system of knowledge has a certain dynamics. There is a great deal that happens concurrently. There is the necessity for various activities to reach some minimum sufficient stage of development in order to support activity at some other level. If things are staged right, and people are in close contact with each other and are highly motivated by effective leadership, then a lot of these things can move rapidly forward together. But remember, there is always a strong element of chance when folks go off exploring. The unfolding of the events depends upon what is discovered, and upon how well the opportunities presented by the discoveries are seized upon and exploited by the overall community of explorers.

### *The Network Community*

Some key resources are required in order to organize such an enterprise. Perhaps the most important capital resource that we drew upon was the computer-communications network, including the communications facilities made available by the ARPAnet, and the computing facilities connected to the ARPAnet at PARC and at various universities. Such a computer-communication network is a really key resource for conducting rapid, large scale, interactive experimental studies.

The networks enable rapid diffusion of knowledge through a large community because of their high branching ratios, short time-constants, and flexibility of social structuring; any participant can broadcast a message to a large number of other people very quickly. It isn't like the phone, where the more people you try to contact, the more time-overhead is added so that you start spending all of your time trying to get your messages around instead of going on and doing something new.

The high social branching ratios and short communications time constants of the networks also make possible the interactive modifications of the systems, all of these systems, while they are running under test. If someone running a course, or doing a design, or creating a design environment has a problem, if they find a bug in the text or the design method, they can broadcast a message to the folks who are leading that particular aspect of the adventure and say, "Hey! I've found a problem." The leaders can then go off and think, "Well, my God! How are we going to handle this?" When they've come up with some solution, they can broadcast it through the network to the relevant people. Thus they can modify the operation of a large, experimental, multi-person, social-technical system while it is under test. They don't have to run everything through to completion, and then start all over again, in order to handle contingencies. This is a subtle but tremendously important function performed by the network, and is similar to having an interactive run-time environment when creating and debugging complex software systems.

There is another thing that happens in the network: it's relatively easy to get people to agree to standards of various kinds, if the standards enable access to interesting servers and services. For example, CIF became a *de facto* standard for design layout interchange because we at PARC said "if you send a CIF file to us we will implement your project". Everybody put their designs in CIF!

We answered our own questions: "Is CIF documented well enough to be propagated around? Does it really work anyway? Does it have the machine independence we've tried for?" That way we debugged CIF and culturally integrated CIF.

Such networks enable large, geographically dispersed group of people to function as a tightly-knit research and development community. New forms of competitive-collaborative practices are enabled by the networks. The network provides the opportunity for rapid accumulation of sharable knowledge. Much of what goes on is captured electronically - designs, library cells, records of what has happened in the message traffic, design-aid software and knowledge - all can be captured in machine representable form, and can be easily propagated and shared.

One reason for the rapid design-environment development during '79-'80 was a high degree of collaboration among the schools. Often, as useful new design aids were created, they were quickly shared. Many of the schools had similar computing environments, and the useful new knowledge diffused rapidly via the ARPAnet.

Another reason for rapid progress was keen competition among the schools and among individual participants. The schools shared a common VLSI design culture; during '79-'80 all used the same implementation system, and batches of projects from the schools were often implemented simultaneously. Therefore, project creation, innovations in system architecture, and innovations in design aids at each of the schools were quite visible to the others. Students and researchers at MIT, Stanford, Caltech, CMU, U.C. Berkeley, etc., could visualize the state of the art of each other's stuff. These factors stimulated competition, which led to many ambitious, innovative projects.

Successful completion of designs, and thus participation in such competition, depended strongly on the quality of the design environment in each school. Therefore, there was strong pressure in each school to have the latest, most complete set of design aids. This pressure tended to counter any "not invented here" opposition to importing new ideas or standards. The forces for collaboration and for competition were thus coupled in a positive way, and there was "gain in the system".

Now, think back to the question, "How do *unsound methods* become *sound methods*?" Remember, you need large scale use of methods to validate them, and to produce the paradigm shifts so that the methods will be culturally integrated. In industry, it's very difficult to take some new proposed technique for doing things and put it in use in a large scale in any one place; a manager trying such things would be accused of using unsound methods. However, in the universities, especially in graduate courses in the major research universities, you have a chance to experiment in a way you might not in industry, a way to get a lot of folks to try out your new methods.

A final note about our methods: The major human resources applied in all of these adventures were faculty members, researchers, and students in the universities. The research of the VLSI System Design Area has often involved the experimental introduction and debugging of new technical and procedural techniques by using the networks to interact with these folks in the universities. These resources and methods were applied on a very large scale in the MPC adventures. There are risks associated with presenting undebugged technology and methods to a large group of students. However, we have found the universities eager to run these risks with us. It is exciting, and I believe that it is appropriate for university students to be at the forefront, sharing in the adventure of creating and applying new knowledge. The student designers in the MPC adventures not only had their projects implemented, but also had the satisfaction of being part of a larger experimental effort that would impact industry-wide procedures.

These experiences suggest opportunities and provide a script for university-government-industry collaboration in developing new design methodologies and new supporting infrastructure in many areas of engineering design. The universities can provide the experimental and intellectual arena; government can provide infrastructure and university research funding; industry can provide knowledge about and access to modern, expensive, capital equipment that can implement experimental designs created by university students and researchers. Modern computer-communications networks, properly used, can tie all these activities together. The implementation of designs closes all the experimental loops.

## 5. Looking Ahead

I wonder where we might apply some of these methods next? Where might some of you apply methods like these in order to aggressively explore new areas? Well, first of all, there certainly are tremendous opportunities further discoveries and evolutionary progress in VLSI design and implementation methodology.

We are now seeing the beginnings of new architectural methodologies appropriate for VLSI in a number of specialized areas of application. For example you might study the work that Dick Lyon is doing to create a new architectural set of "VLSI building blocks" for bit-serial digital signal processing.<sup>21</sup> Wouldn't it be interesting if those techniques could now be tried in a few courses? We'd find out if people can really learn about signal processing with VLSI, and then quickly compose working systems, thus providing a reality test of Dick's ideas.

There are many other areas of digital system architecture ripe for the introduction of new architectural methodologies appropriate for VLSI. There are areas like computer graphics for providing high-bandwidth visual displays for interactive personal computing systems, and the generation of computer images for electronic printing and plotting. There's image processing, taking digitized input image data and processing it to recognize and detect things, with applications in OCR systems, visual input systems for controlling robots, smart visual sensors for various defense systems, that sort of thing. There are areas like data encryption and decryption. So there's a whole world of specialized architectural areas that people can now explore, given that they have access to a VLSI design environment and to quick turnaround implementation to try out their ideas. As successes accumulate, the underlying knowledge and the detailed design files can be rapidly propagated around the VLSI network community.

There are many opportunities for evolving new design and analysis aids appropriate for the new design methodology. Progress has been rapid so far,<sup>19</sup> but there is plenty more to do. Those interested in creating and testing new design aids might ask yourselves "What can I create and then introduce over the network that would be valuable to the VLSI community, that might integrate with the overall activity?" That line of thinking, taking into account the current state of the community, and the means of introducing new ideas into the community for testing and validation, may increase your chances of successfully creating something that becomes culturally integrated.

For example, the early circuit extractor work done by Clark Baker<sup>16</sup> at MIT became very widely known because Clark made access to the program available to a number of people in the network community. From Clark's viewpoint, this further tested the program and validated the concepts involved. But Clark's use of the network made many, many people aware of what the concept was about. The extractor proved so useful that knowledge about it propagated very rapidly through the



community. (Another factor may have been the clever and often bizarre error-messages that Clark's program generated when it found an error in a user's design!)

Another area of opportunity is in the evolution of standards. For example, we need a standard "process test chip" for the back-end foundry interface, so that designers and foundry operators will have a mechanism for deciding to shake hands and exchange dollars for wafers. Although some strawman versions have been proposed, there is no standard now. Perhaps a standard process test chip could be evolved by inserting strawman versions into wafers that are run for university multiproject chips sets. The community could then gradually converge on a workable standard.

There are opportunities for further evolution of implementation systems. Also, similar design and implementation methods could be mapped into technologies other than nMOS. Design primitives, design rules, and design examples could be created, for example, for CMOS and then run through the same kind of scenario as above to introduce those into a university community.

I myself have become interested in the prospects for bringing about a convergence of the work in VLSI design methodology with work based in knowledge engineering.<sup>22,23</sup> There is the possibility of creating knowledge-based expert systems to aid VLSI system designers. I can imagine directing the evolution of such expert systems by using similar methods to those described above: trying out ideas, prototyping them, evaluating them, and bringing them in large-scale use within a computer-communication network community. But an added twist is possible here, that of making knowledge about expert systems accessible to the larger CS community, a community now knowing about VLSI. That way we could help to generate a common literacy about knowledge, a common knowledge representation language, and knowledge about the methods of knowledge engineering.

You'll note that the experimental methods described in this talk aren't limited to application in the exploration of microelectronic system design. I find it fascinating to think about applying these methods to the rapid exploration of other domains of engineering design that may be operating under new constraints, and thus be full of new opportunities.

For example, it is becoming common in some industrial environments for folks to do mechanical system design by using computers to specify the shape and dimensions of parts and to generate the tapes for numerically controlled machine tools that can implement the parts. Consider the opportunity here: What if we documented a simple design method for creating mechanical systems under the assumption that the parts are to be remotely machined and assembled in some sort of "magical automatic factory". Then ask the question, "Well, how would you teach mechanical design under the many new constraints imposed by the remote factory?" If you had access to such a factory, or if you could even emulate it using manual procedures where necessary, you could put in place the same sort of overall experimental environment to develop from very early crude principles some sort of new design methodology that would be appropriate for that environment. In that way one could evolve an entire design culture of methods, courses, design examples, design aids, etc., using the methods described above, and that culture could be rapidly spread out through the networks into a large university community.

I am very interested in studying and experimenting further with techniques for creating, refining, and culturally integrating new engineering design methodologies. If any of you folks engage in similar work, especially within the university computer-communications network community, I'd be very interested in learning of your experiences. I'd enjoy brainstorming with you on how to improve the underlying methods, and how to spread knowledge about the results.

## 6. Acknowledgements and Conclusions

I am deeply indebted to many people for their contributions and help in creating the design methods, the textbook, and the implementation methods and system, and also the university VLSI design courses, design environments, and research programs. There are literally hundreds of people who have played important roles in the overall activity. Students, researchers, and faculty members in the universities, and a number of industrial researchers, industrial research managers, and government research program managers have been actively involved in these events. I am at a loss to acknowledge all of the individual participants.

However, I would like to individually acknowledge some some folks at PARC who've worked on this research since the early days. I am thinking of Doug Fairbairn, who was with us during the key early years; Dick Lyon, who has contributed so much to the effort; Alan Bell and Martin Newell for their innovations and efforts in the creation of VLSI implementation systems that have supported so well the validation and spread of VLSI knowledge. I'd especially like to acknowledge the support and encouragement that all of us at PARC have received over the years from the senior research management of Xerox Corporation, in particular, from Bert Sutherland.

Let's look at the photo of Alan Bell again (Fig. 3), and think back to the MPC79 effort. I'm sure you now sense that MPC79 was not just a technical effort, that there was a tremendous human dimension to the project. So many folks were simultaneously creating and trying out things: students and researchers trying out new designs that were very, very important to them; instructors and project lab coordinators trying out the new courses and project lab facilities; at PARC the new implementation system was coming into existence, under the pressure of trying to provide VLSI implementation service to the many university designers. This built up into a tremendously exciting experience for all participants, a giant network adventure that climaxed as the design-cutoff time approached, and the final rush of design files flowed through the ARPAnet to PARC.

So when you see someone interacting with a personal computer connected to a network, rather than jumping to the conclusion that you are observing a reclusive hacker running an obscure program, you might ask yourself "I wonder what adventures this person is involved in?" Remember, you may be observing a creatively behaving individual who is participating in, or perhaps even leading, some great adventure out in the network!

These events are reminiscent of the pervasive effects of the telegraph and the railroads, as they spread out everywhere during the nineteenth century, providing an infrastructure people could use to go on adventures, to go exploring, and to send back news of what they had found. I think of personal computers and the computer communication networks as a similar sort of infrastructure, but here and now, as we explore the modern frontier - - the frontier of what we can create.

The new knowledge and products our VLSI design community is creating will have tremendous social impact, by helping rapidly spread and increasing the power of the new personal computing and computer-communication infrastructure.

Thus your work in computer science and VLSI system design is expanding the opportunities for all of us to go on all sorts of grand adventures in the future!

## REFERENCES

1. C. Mead and L. Conway, *Introduction to VLSI Systems*, Limited printings of prepublication drafts of a text in preparation, Xerox Palo Alto Research Center (PARC), Palo Alto, CA; (a) Chapters 1-3, September 1977; (b) Chapters 1-5, February 1978; (c) Chapters 1-9, July 1978.
2. R. Hon and C. Sequin, *A Guide to LSI Implementation*, Limited Printing, Xerox PARC, September 1978.
3. G. Steele, Jr. and G. Sussman, *Design of LISP-Based Processors or, Scheme: A Dielectric LISP or, Finite Memories Considered Harmful or, LAMBDA: The Ultimate Opcode*, AI Memo No. 559, Artificial Intelligence Laboratory, M.I.T., March 1979.
4. C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, MA, 1980.
5. R. Hon and C. Sequin, *A Guide to LSI Implementation, 2nd Ed.*, Xerox PARC Technical Report SSL-79-7, January, 1980.
6. L. Conway, *The MIT '78 VLSI System Design Course: A Guidebook for the Instructor of VLSI System Design*, Limited Printing, Xerox PARC, Palo Alto, CA, August 1979.
7. D. Fairbairn and R. Lyon, "The Xerox '79 VLSI Systems Design Course", *Xerox PARC Videotapes and Lecture Notes*, Xerox PARC, Palo Alto, CA, February, 1979.
8. L. Conway, A. Bell, M. Newell, R. Lyon, R. Pasco, *Implementation Documentation for the MPC79 Multi-University Multiproject Chip-Set*, Xerox PARC Tech. Memorandum, 1 January 1980.
9. J. Clark, "A VLSI Geometry Processor for Graphics", *Computer*, Vol. 13, No. 7, July, 1980.
10. J. Holloway, G. Steele, Jr., G. Sussman, A. Bell, *The Scheme-79 Chip*, AI Memo No. 559, Artificial Intelligence Laboratory, M.I.T., January 1980.
11. L. Conway, A. Bell, M. Newell, "MPC79: The Demonstration-Operation of a Prototype Remote-Entry, Fast-Turnaround, VLSI Implementation System", *Conference on Advanced Research in Integrated Circuits*, M.I.T., January 28-30, 1980.
12. L. Conway, A. Bell, M. Newell, "MPC79: A Large-Scale Demonstration of a New Way to Create Systems in Silicon", *LAMBDA, the Magazine of VLSI Design*, Second Quarter, 1980.
13. T. Strollo, et al, *Documentation for Participants in the MPC580 Multiproject Chip-Set*, Xerox PARC Technical Memorandum, 7 July 1980.
14. R. Rivest, "A Description of a Single-Chip Implementation of the RSA Cipher", *LAMBDA, the Magazine of VLSI Design*, Fourth Quarter, 1980.
15. D. Fairbairn, R. Mathews, J. Newkirk, et al, *Videotape VLSI Design Course based on the Mead-Conway text "Introduction to VLSI Systems"*, VLSI Technology, Inc. (VTI), Los Gatos, CA, 1980.
16. C. Baker and C. Terman, "Tools for Verifying Integrated Circuit Designs", *LAMBDA, the Magazine of VLSI Design*, Fourth Quarter, 1980.



17. R. Bryant, "An Algorithm for MOS Logic Simulation", *LAMBDA, the Magazine of VLSI Design*, Fourth Quarter, 1980.
18. A. Bell, "The Role of VLSI Implementation Systems in Interfacing the Designer and Fabricator of VLSI Circuits", *Proc. of the International Telecommunications Conference*, Los Angeles, Nov. '80.
19. L. Conway, "University Scene", *LAMBDA, the Magazine of VLSI Design*, Fourth Qtr., 1980.
20. J. Clark and M. Hanna, "Distributed Processing in a High-Performance Smart Memory", *LAMBDA, the Magazine of VLSI Design*, Fourth Quarter, 1980.
21. R. Lyon, "Signal Processing with VLSI", *Limited printings of lecture notes for a "constantly evolving talk"*, Xerox PARC, 1980.
22. E. Feigenbaum, "The art of artificial intelligence - Themes and case studies of knowledge engineering," *Proc. of the 1978 National Computer Conference*, AFIPS Press, Montvale, N.J., 1978.
23. M. Stefik, et al, "The Architecture of Expert Systems: A Guide to the Organization of Problem-Solving Programs," to appear as Chapter 3 in: F. Hayes-Roth, D. Waterman, D. Lenat, (Eds.), *Building Expert Systems*, (a textbook in preparation).

#### SUGGESTED READING REFERENCES

- H. Simon, *The Sciences of the Artificial*, The M.I.T. Press, Cambridge, MA, 1969. (2nd Ed. in Press).
- T. Kuhn, *The Structure of Scientific Revolutions*, 2nd Ed., Univ. of Chicago Press, Chicago, 1970.
- L. Fleck, *Genesis and Development of a Scientific Fact*, F. Bradley and T. Trenn, Translators, T. Trenn and R. Merton, Editors, University of Chicago, 1979. (Originally published as *Entstehung und Entwicklung einer wissenschaftlichen Tatsache: Einfuhrung in die Lehre vom Denkstil und Denkkollektiv*, Benno Schwabe & Co., Basel, 1935.)
- C. Levi-Strauss, *Mythologiques*, Vols. I-IV, Plon, Paris, 1964, '66, '68, '71.
- H. Garfinkel, *Studies in Ethnomethodology*, Prentice-Hall, Englewood Cliffs, N.J., 1969.
- B. Latour and S. Woolgar, *Laboratory Life: The Social Construction of Scientific Facts*, Vol. 80, Sage Library of Social Research, Sage Publications, Beverly Hills, 1979.
- D. Crane, *Invisible Colleges: Diffusion of Knowledge in Scientific Communities*, Univ. of Chicago Press, Chicago, 1972.
- D. Englebart, R. Watson, J. Norton, *Advanced Intellect-Augmentation Techniques*, Stanford Research Institute, Menlo Park, CA, 1972.
- J. Licklider and A. Vezza, "Applications of Information Networks", *Proceedings of the IEEE*, Vol., 66, No. 11, November, 1978.
- J. Lederberg, "Digital Communications and the Conduct of Science: The New Literacy", *Proceedings of the IEEE*, Vol., 66, No. 11, November, 1978.



## MOSIS -- THE ARPA SILICON BROKER<sup>1</sup>

Danny Cohen and George Lewicki

USC/Information Sciences Institute

This paper is actually an edited transcript of the talk presented at the conference. Many references to visual accompaniments, difficult to reproduce here, have been eliminated.

### INTRODUCTION

The idea of a silicon broker was conceived by Carver Mead some time ago as a way to give a large community of chip designers access to fabrication services and as a way to speed up the fabrication process. MOSIS is the ARPA silicon broker that we have implemented at ISI. With MOSIS we are trying to totally isolate the designer from all the trivia that fabrication requires.

The main objective of ISI's VLSI project is to support the fast turnaround requirement of the ARPA VLSI community and of related programs. Another of our objectives is to help expand the VLSI design community by supporting research institutes and universities that are actively involved in VLSI. We hope to help MIT, Caltech, Berkeley and other universities train as many VLSI students as they can.

In addition, we'd like to encourage more vendors to offer custom VLSI services. We were pleasantly surprised at the number of organizations already in the business of offering those services.

For the time being, we are sorry to report we can serve only the ARPA VLSI community. However, other government-sponsored users may gain access to MOSIS by special arrangement with ARPA. If you are interested in our service and your project is government sponsored, please contact us or ARPA, and we will try to help you. Remember that NSF is part of the US Government, so people sponsored by NSF will probably be able to participate.

<sup>1</sup>This research is supported by the Defense Advanced Research Projects Agency under Contract Nos. MDA903 80 C 0523 and MDA903 81 C 0335. Views and conclusions contained in this paper are the authors' and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government, or any person or agency connected with them.

MOSIS

The MOSIS system developed from an idea demonstrated in recent years by Caltech and by the MPC project at Xerox PARC. We'd like to acknowledge everyone who helped us, but this is only a partial list. We would like to mention Carver Mead--he was the first to put together many chips, and he taught us how to do it; and Lynn Conway, whom you have just heard telling us all about the MPC project at Xerox PARC. Thanks also to the fantastic crew there: Alan, Martin, Dick, Ted, and many others--it's impossible to name everyone here; please accept our apologies.

ISI's silicon broker works as follows: Users who have obtained access to MOSIS communicate directly with the system via electronic mail. Most users are on the other side of the ARPANET, whether across town or across the country. MOSIS understands various types of information, such as, "this is a description," "this is a pad," and "this is the technology we want," and it knows that CIF files describe the geometry of a project. MOSIS accepts several types of requests, for example, "please start a new project." All requests are very formal, because machines, not people, read them. Questions about sending requests to MOSIS can be sent to MOSIS@ISIF. The questions should be stated in plain English, e.g., "Please tell me what to do." The answers from us will probably be equally cryptic: "We couldn't understand your message, but if you want to talk to us, do such and such and we will send you the MOSIS User's Manual." Save time and trouble by reading the MOSIS User's Manual--it explains everything a user needs to know.

All user-provided information flows through MOSIS to MrBill, our geometry handler that checks CIF files, packs sets of projects onto a (smaller) set of dies, translates each die into MEBES format, makes bonding diagrams, and more. Ron Ayres wrote MrBill in ICL--beautiful language, beautiful system, works magic, very efficient. For example, it can plot CIF files like Figure 1. Figure 2 is a slightly more complicated plot. It's not clear exactly what MrBill drank before he plotted it, but we were told, it's OK, it's a bubble memory. MrBill's primary task is to produce tapes that the foundry uses to make masks.

After MrBill does his work, the next step in the process is mask fabrication. Mask houses expect two types of things from us: tapes with MEBES files and job decks. MEBES files contain the information that the mask house uses to make bitmaps (which are made into masks). A job deck, about one percent of the size of a MEBES file, maybe less, contains the specifications for each MEBES file--parity, record size, etc.

Fabrication itself is very simple because somebody else does it. Once the masks are made, all we have to do is drive three, four, or maybe ten miles in Silicon Valley with the masks to a wafer fabricator. (It is wise to drive slowly to make sure the masks don't break.) After that, if we're lucky--and typically we are--we end up with a couple of wafers.

Once we have the wafers, we like to probe each of the chips, not just all the wafers, so that no one will tell us later, "Maybe only the north part of

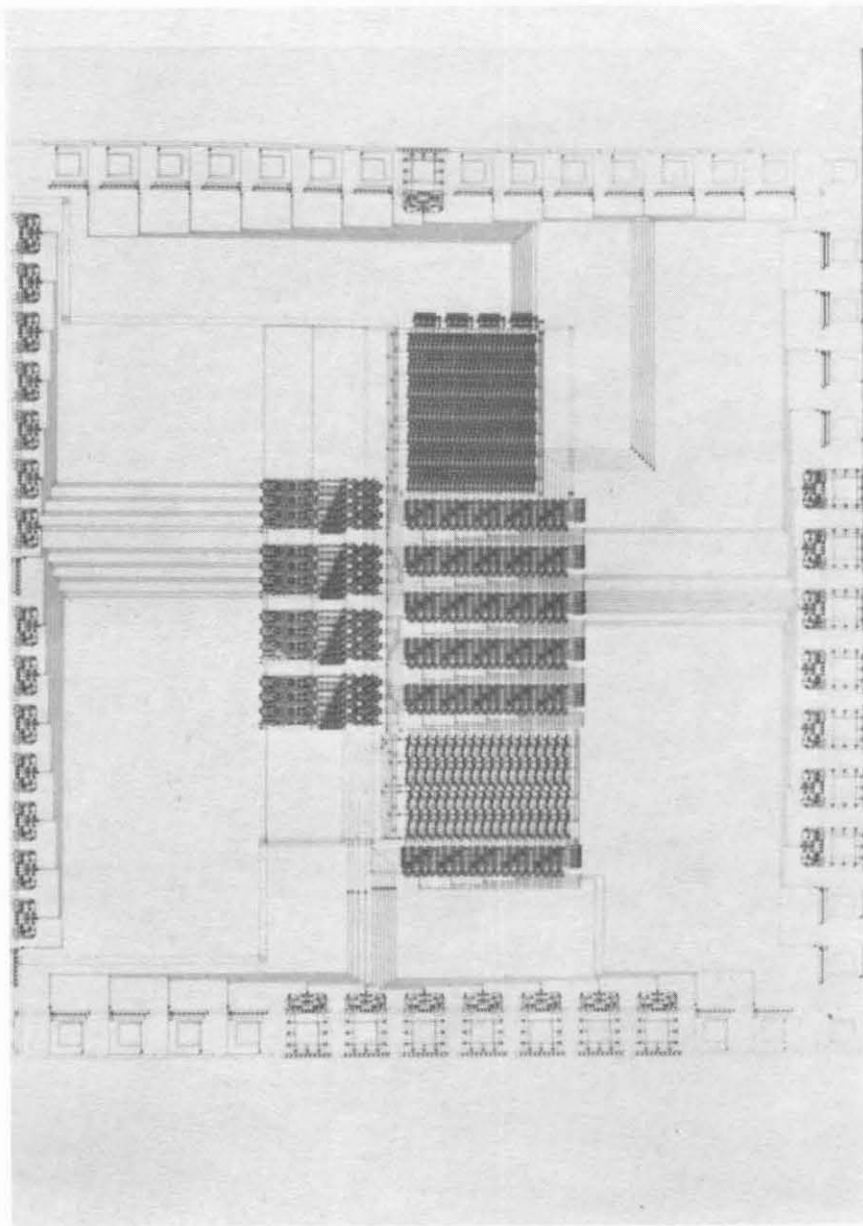


Figure 1: Simple plot produced by MrBill from a CIF file

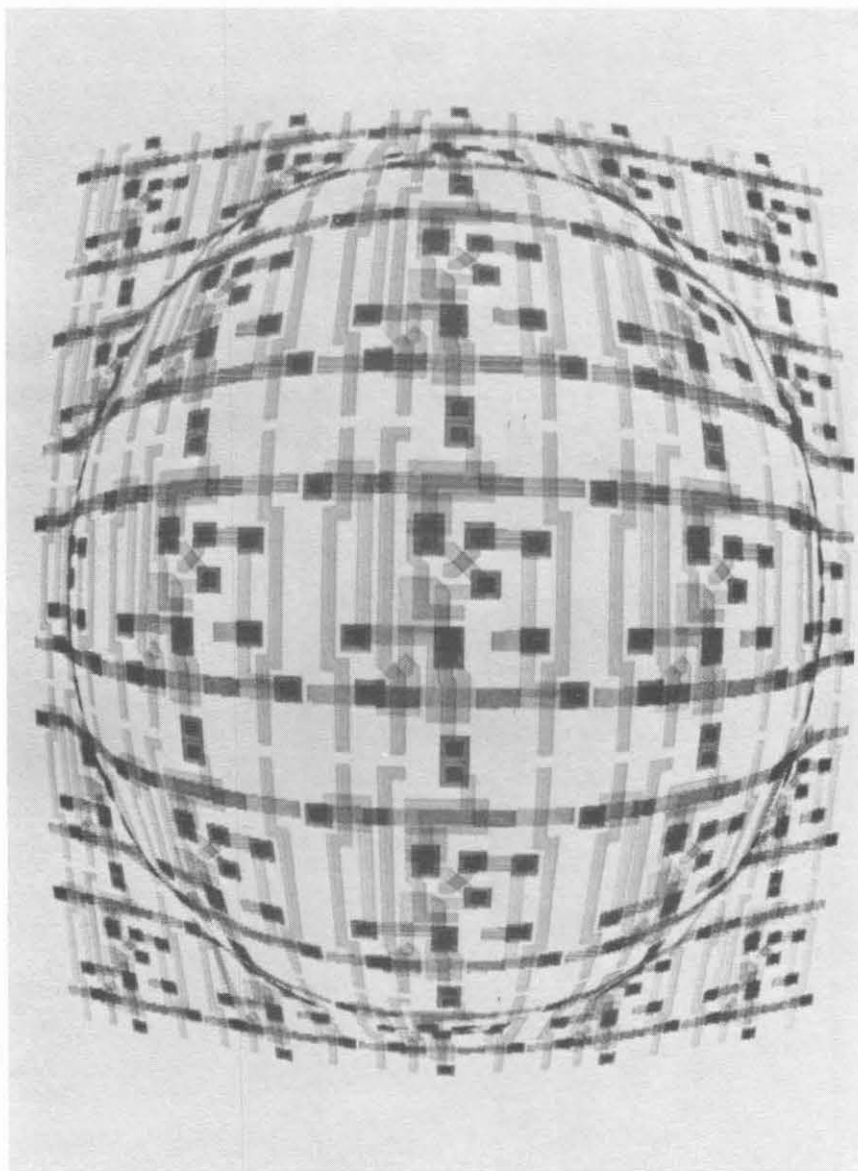


Figure 2: A more complicated plot of a bubble memory

the wafer is good; if you happen to have a southeast project on it, you probably lose." We like to make sure that the wafer is uniformly good.

Then we break the wafers into individual chips, package them, and run some more tests, if we can. Afterwards we distribute the chips to the users, and each user examines his chips, points fingers at everyone, resubmits, etc.

#### STANDARDS FOR MOSIS

We prefer continuous-spooling mode, which means that we don't like to advertise deadlines. Whenever we have enough projects, we fabricate them. The sooner you submit, the sooner your design will be fabricated.

For the time being, we support CIF 2.0+. If you know only about CIF 2.0, fine; we support it. There are several other features we support, and perhaps we'll eventually convince the entire community to use them.

At present, we support nMOS with the Mead-Conway design rules. More processes will eventually be offered. This means that we do not now support 2-layer metal, buried contacts, etc., but we will later. Currently we use  $\lambda$  equals 2.5 microns. This is a feature size of five microns. We have talked to several people about smaller feature size, and we are in various stages of negotiations about smaller values.

Once given a file, we can change  $\lambda$  a little bit, but not a lot. If a big change of  $\lambda$  is necessary, the designer has to make the changes. If the design is for 2.5 and we have 2.0 available, we might change the size if the designer allows us to.

Our standard packages for bonding projects are 40 and 64. If we can find packages for 89 bond projects, we might be able to bond them, but it might take more time.

We try to provide fast turnaround by streamlining all the interfaces. We have been told by industry that if we pay a premium, we can get faster services. We are not sure this is what we want to do right now. With more money we know we can get faster service. We are trying to see how fast service is if we know about tape parity, registration marks, CDs, and all the other details that are required for fabrication.

#### PROBLEMS

We are constantly trying to improve the service from MOSIS. We like to try new software for its added features. We like to try new mask houses so we don't have to depend on one source. We like to try many fabrication lines. We like to change the way we test wafers, packaging, etc.

We have problems in the process of qualifying any changes, that is, making sure a change is really an advance. For example, the first problem we had was



deciding how to qualify a new set of software. The problem arises in comparing two masks, one produced by the old and the other produced by the new system. Are the patterns really the same? A microscope is supposed to help, but it can't do a good job. We tried many ways and finally worked out a very strange technique. Suppose you want to compare mask A and mask B. What we did was to overprint A and B bar [the reverse of B], and A bar and B. In this way we discovered all the changes. We did all the printing on one plate so we wouldn't have to use a special microscope.

We applied this test after we were sure there were no other problems. We were shocked by what we discovered—one little bug that turned out to be many bugs in the manuals, not a bug in the software. But it was just as bad, so we had to fix everything until we finally got all those squares to be exactly the way they were supposed be, blank.

We have also had problems in the preparation of a job deck. A job deck is a definition of a wafer. Figure 3 shows a wafer containing 18 different die-types. Each die-type requires six files, so over 100 files are involved in preparing this wafer. A lot of coordinating is therefore necessary, and we would like to make sure everything happens right.

Some of the companies we've worked with (Xerox, Boeing) share horror stories with us about the production of an accurate job deck. Our goal is to generate job decks with computers. Figure 4, for example, is an input for a program that generates a job deck (unfortunately, not for the wafer shown in Figure 3. Sorry about that.). The input contains the name of the run, like N11E, and the definition of each layer. The letters D and C determine the dark or clear mask—very important, or else you get some odd results. The name of the level and the name of the job have to be written correctly so the fabricators do not make mistakes. The input also contains some coordinate, and the map, which controls the position and the choice of over 100 files. With all of these variables, there is high potential for something to go awry.

We have to screen new fabricators carefully. Ideally we'd like to give them a form to fill in, and then we would continue from there, if their qualifications were close to what we expected. In reality we ask the fabricators, "What technology do you offer? nMOS? CMOS? What?"

We also ask, "What are your design rules?" Actually, we don't ask, "What are your design rules?" We say, "Those are OUR design rules. Do you support them?" And sometimes we get answers, "Yeah, we support them. Lambda equals, say, two microns, for everything except ..." and we say, "Too bad. You don't really support our design rules for this feature size...." We have to decide at what feature size our design rules are supported.

Next we ask the potential fabricator for electrical parameters, everything we need to know about masks, polarity, bloating, etc. Then we tell the fabricator how we like to measure. As a matter of fact, selecting a fabricator is not quality control—it should be a process control, insurance that everything meets our standards, including turnaround time, and, obviously, the expense.

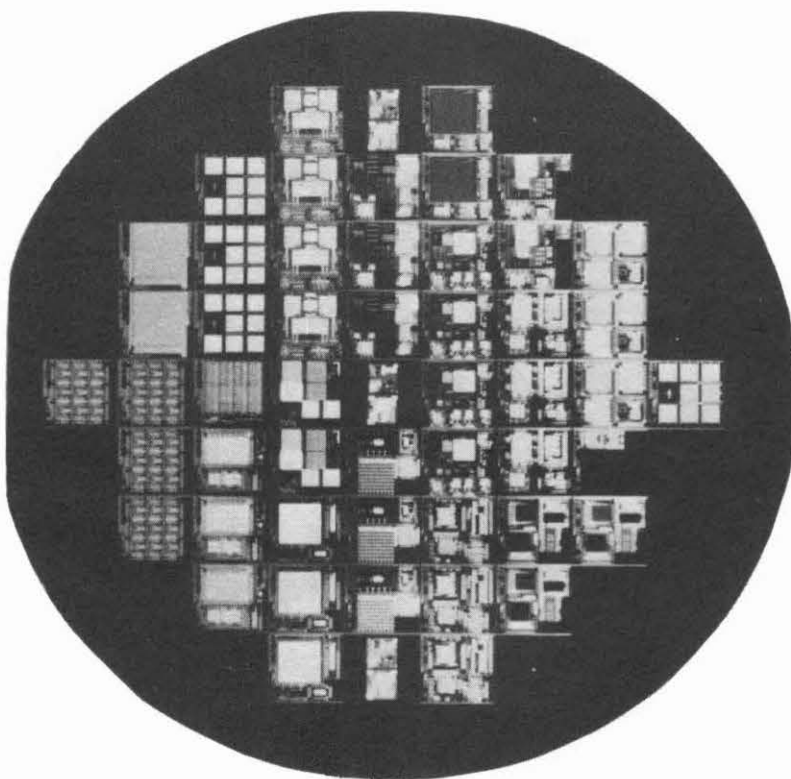


Figure 3: Wafer containing 18 different die-types

# Job Deck for Masks

M11E

01D ND DIFFUSION

02C NI IMPLANT(DEPL)

03D NP POLY

04C NC CUT(CONTACT)

05D NM METAL

06C NG OVERGLASS

24000 22400 6700 7100

G  
 C D E F H  
 I J K L M N O  
 A Q P O A D Q I A  
 B O C G B E P J B  
 L M H Q C F N K M  
 I J P H L G O  
 K D E F P  
 N

Figure 4: Example input for a program that generates a job deck

We specify five electrical parameters to the fabricators. We ask that the enhancement devices' threshold voltage be about +0.8V. For depletion, it should be about -3.5V. For  $k=4$  and for  $k=8$ , minimum size inverters  $V_{inv}$  should be about +2.0V, and the poly sheet resistance should be less than 50 ohm/square.

Obviously, this process control doesn't cover everything. Our expectations are that, when we go to reputable lines, they know how to do the things we have to have done in order to support our designers and they will produce good chips all the time. We would not be surprised if one day we find wafers that don't do anything right but that do meet these five criteria. When this happens, we are not going to sue anyone--just say, "Sorry! We are not going to use this line anymore." We are not interested in finding out exactly why something went wrong.

### TESTING

Next, we have the issue of testing. We don't need tests that are designed to calibrate fabrication lines because we don't care to calibrate fabrication lines. There are already people whose job it is to calibrate the lines. The tests that are important to us tell us something about what yield or what performance users can expect with our design rules; those areas are our concern. We would love to have standard industry wafer-acceptance procedures. We'd like to be able to design one test chip into every wafer, accept the produced wafer, and then test it. If it passes the test, we say it's a good wafer, and we pay. If it does not, we don't take that wafer. We'd like to establish a standard that will be accepted both by industry and, obviously, by users.

Unfortunately, we have not reached that point yet. We are working on a standard. JPL, Xerox PARC, NBS, and the Integrated Circuit Lab of HP are participating in this effort. We have made some progress, but, again, we are not there yet.

First of all we are trying to test the basic elements, like transistors. Then we like to test the building blocks, like inverters, to see if they work to our specs, and then even more complicated random fault structures. In order to do that, we have our "standard" test patterns, which are designed for probing, not for bonding.

That test vehicle is an interesting camel. It was supposed to be a lion. The committee that designed it met too many times. It's very complicated. It has had many tests, and we are trying to simplify it. Maybe we will be able to turn it into a lion again. But we don't know yet. We will work on it.

Another issue is how to verify the completeness of the testing. We would be uncomfortable in a situation where all the tests are passed with flying colors, and no device works, or most devices don't work.

In the past, there have been two situations in which tests were perfect but devices didn't work. When that happens you say, "Gee, that's too bad. Let's change the test." We don't want to go into too many details, but let us describe one of those cases to you.

On a certain run we had, among others, die D and die E. Unfortunately, an error crept into the job deck when someone at the mask shop decided to retype it manually. When the person retyped it he interchanged the diffusion level of the two dies.

However, our test patterns on the two dies are identical! When we probed, everything was perfect! So now we say, "Aha! If we put the die designation on all layers..." (by the way, we thought about doing that before but never did or the error would have been caught). Now we know how to find this error. But we have no way of knowing how many other problems will pass all our tests without being caught.

We use both small test patterns that are part of every die and a few bigger drop-in tests. All the test patterns of all dies on all wafers are probed. We actually probe every die. We compute the mean and standard deviations over the sample of 46 to 50 dies. We are looking for some interesting patterns, but we hope never to find them.

Two- and three-dimensional analyses of problem data do not reveal any significant inter- or intra-wafer pattern. Two-dimensional analysis, for example, indicates whether the north part of the wafer is better than the south or whether the middle is much better than the edge. Three-dimensional analysis shows up a difference between wafers. Maybe one wafer is OK, and other wafers are not. We compute both by wafer and by position, and by many other statistical means.

We have been very delighted not to find significant patterns. If we found significant patterns, for example, that the northeast corner is always the best, we would be flooded with requests from users: "Please put my job on the northeast," or "put mine on wafer number three," or something like that. We believe that Monday wafers are not really as good as Tuesday wafers, but we cannot prove it!

We have also experimented with several comprehensive structures that test typical user devices. Years ago, there was a notion of a typical picture for computer graphics. A thousand lines was considered a typical picture, and everyone was supposed to support such a typical picture. What we need now is a typical user device that we can put on every wafer, try it, and if it works, then everything is OK. If it doesn't work, we have a problem. We are still looking for such a device.

As a matter of fact, we are designing several canaries<sup>2</sup> just for this purpose. One of our canaries is a 19-stage ring oscillator that Xerox's MPC used. We'd like to use it as many times as we can. One of the neatest things about this ring oscillator is that it uses only three pins. This is very important, because we can nearly always bond it in addition to other projects that don't require all these pins. When we bond projects we always try to bond the 19-stage ring oscillator and to test all of the oscillators to see that everything works.

In addition, we are trying to get some yield information from the world's slowest 4K RAM. We try to come up with ratios such as 3 bits out of 4K didn't work on so many units to see if we can derive some yield statistics that are meaningful for users. Anyone who has entries or suggestions for this collection of canaries is most welcome to submit them for consideration.

Thanks are due JPL and NBS for providing the following test structures. Figure 5 shows die F of run B; what is evident here are some random fault structures. There are several miles of metal over poly, etc. One of the interesting things to see here is that this is both a drop-in as well as a user device (at the lower right-hand corner). We don't really make any differentiation between user projects and drop-ins.

There were random fault structure dies with more miles of step coverage, and their logarithmic connections were visible. With random fault structures like this, there is always the hope that the small portions of the structure are small enough not to have any faults and the big portions are large enough to make it easy to find the faults. And the worst that can happen is that all of them fail or none of them fails. Then you know you are looking at the wrong range.

We had another interesting drop-in from NBS. It is interesting because several test patterns are repeated with variable geometry. It is revealing to learn more about the geometry and compare it with claims made by manufacturers.

#### PACKAGING

Our standard packages, as mentioned earlier, are 40 and 64 pins. We might bond several projects in the same package if they go to the same customer. It often happens that several small user projects can be bonded together. We always try to bond as many test structures as we can; for the time being this includes only the 19-stage ring oscillator. If we have more later, we will try to bond them too, but never at the expense of the paying passengers.

<sup>2</sup>Canaries used to be employed in underground mines as indicators of air quality. If the air was bad, the canary would die, but the miner would have a chance to return to good air.

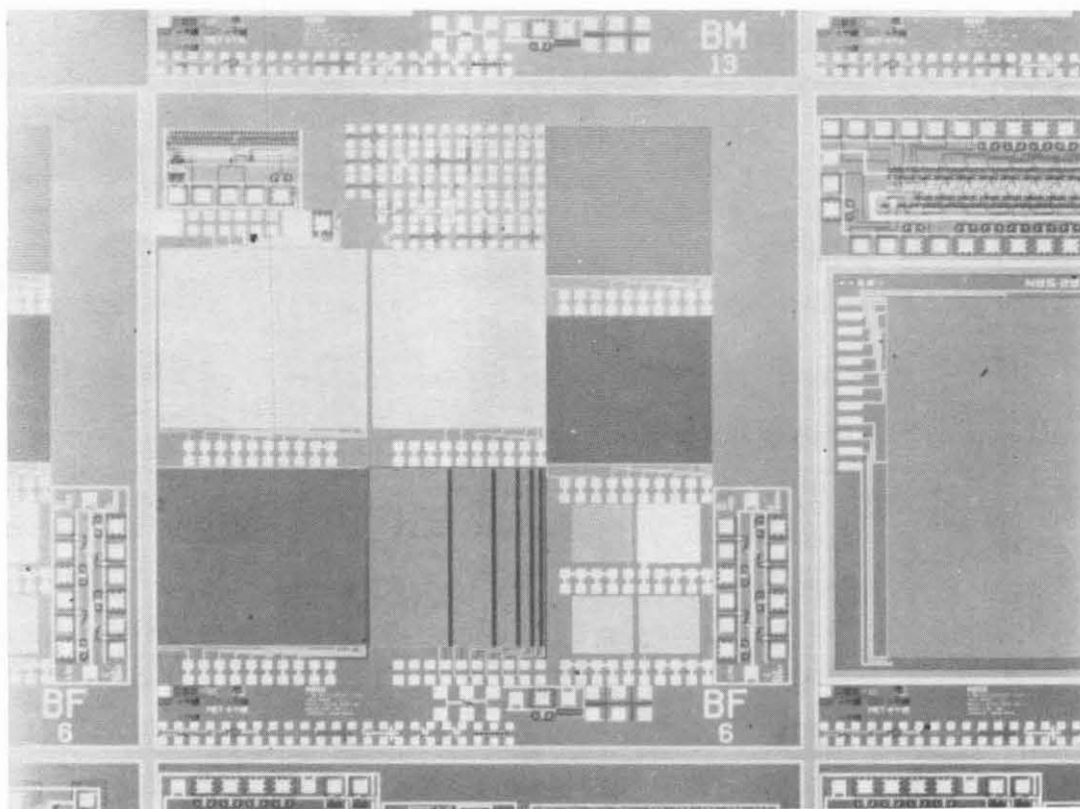


Figure 5: Die F of run B



Researchers at Lincoln are pursuing several techniques for bonding standardization without imposing limitations on the designers. At present we bond manually. We'd like to be able to go to automatic bonding, but the large variation between projects prevents that, at least at reasonable cost. The folks at Lincoln are trying to work out some techniques that will enable us to bond automatically.

Our strategy for die distribution is as follows. We try to fabricate  $n$  copies<sup>3</sup> for each project, such that the probability of giving a designer at least two dies that are mask defectless and silicon defectless will be greater than 90 percent. Now maybe 90 is not a high-enough number, but 90 will have to suffice. In the worst case we can always refabricate the die just a few weeks later. The die should also be bonding defectless. If the bonding is not 100 percent, there is no point in having a perfect project. From time to time we have discovered some problems in bonding.

Using the available data, we were able to achieve our goal of 65 projects on 18 die types with one wafer set. We showed this wafer set before (Figure 3). This die J (Figure 6) happens to have eight different projects on it. Some of them could be bonded in the same package (utilizing unused pins), some of them not; the arithmetic of how many of each can become very interesting.

One of the most interesting projects was done by a student of Chuck Seitz, Eric Barton. Eric was very impressed by Chapter 7 in Mead and Conway, and he decided to do a self-timed project. So he had his own clock wired into the project; it can be seen in Figure 7. When power is applied, the hands actually move. We disconnected it at 8:00 this morning so it still registers that time. The clock was a great thing. We never knew about it until we looked through the microscope. First time we saw it under a microscope we checked--it was three minutes slow.

## CONCLUSION

We want to push lambda but not at the expense of the design rules. We would like to see if we can really reduce the design rules. It's nice to say one micron, but obviously we do not have to stop there. When people from industry come to us with submicron processes, we will be delighted to check each of the processes. We're always ready to add more features to nMOS and always willing to use other technology. Though we're not sure in exactly which order...

By the end of 1981 we expect to support over a thousand designers and a thousand projects. We want to be very careful with this kind of prediction. We like to think that we are underestimating: we would like to see more users. Please, feel free to try us; we hope that we will be able to accommodate most of you.

---

<sup>3</sup>Needless to say, this magic number  $n$  depends on the active area of the project.

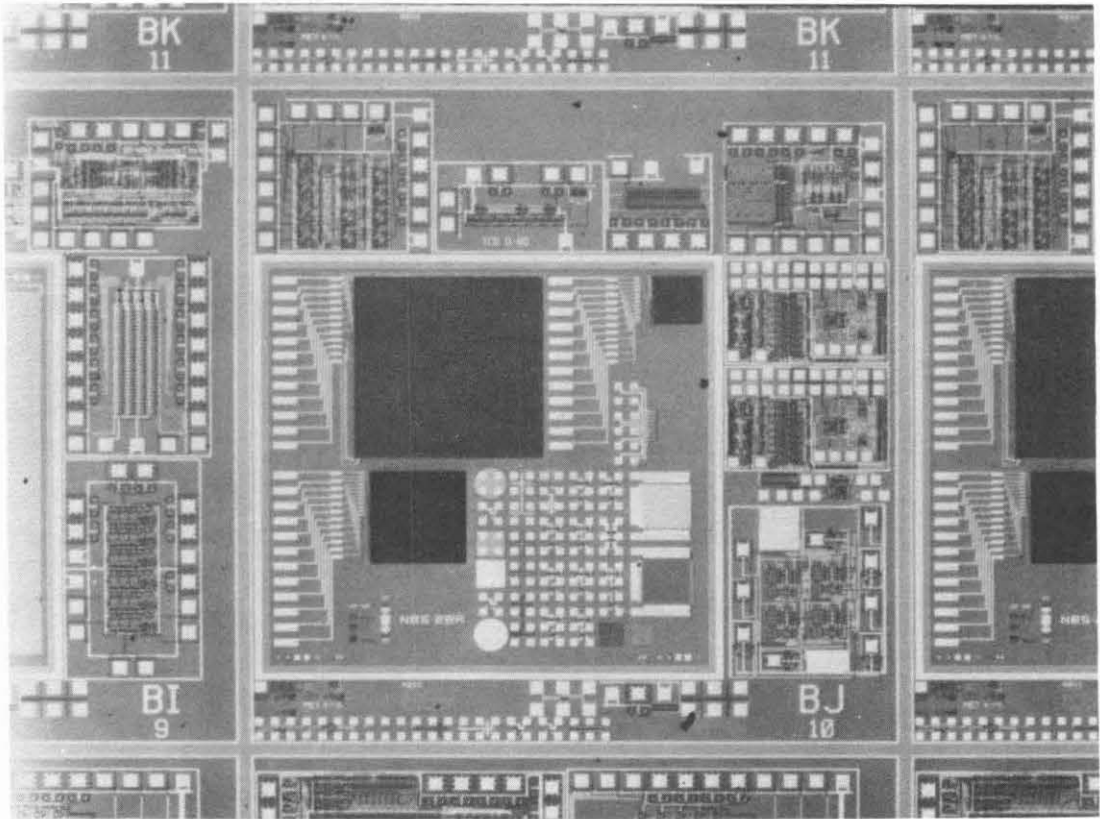


Figure 6: Die J with 8 different projects

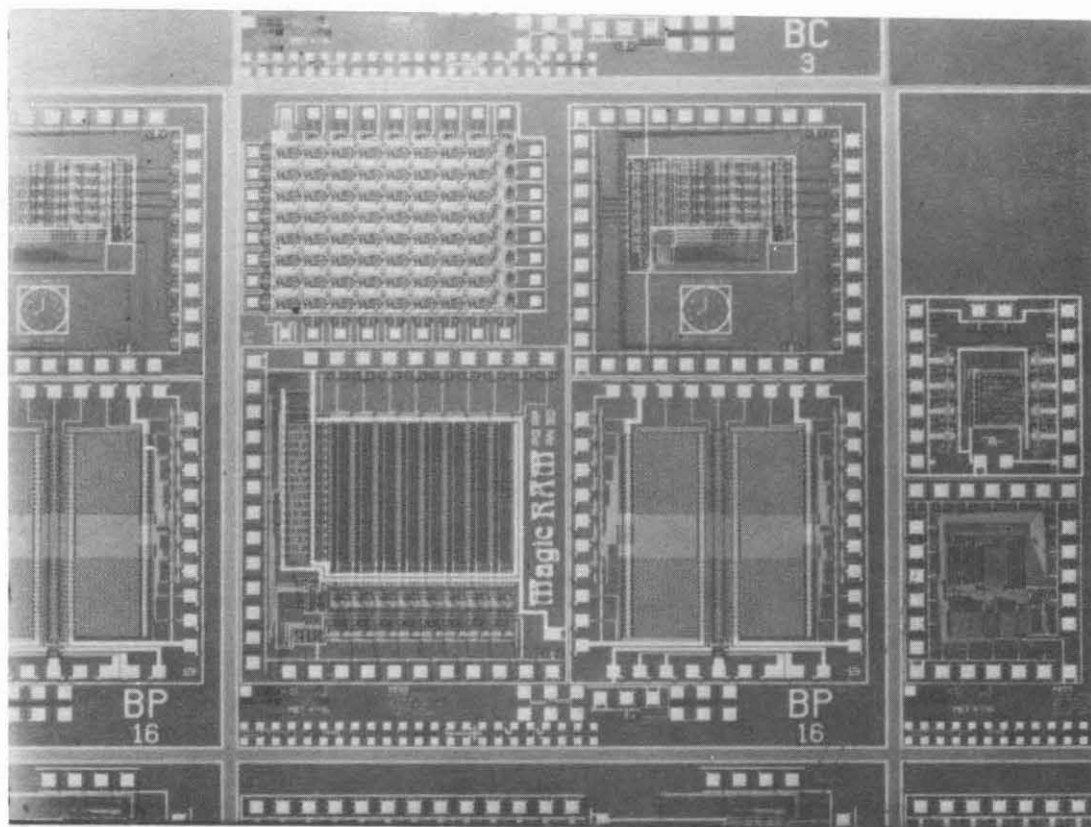


Figure 7: A self-timed project

ACKNOWLEDGMENTS

Thanks to ARPA, which provides all the funds for this project; JPL and NASA, which helped us a lot in the first run; NBS, PARC, and the HP Integrated Circuit Lab, which has helped us a lot in the testing. Thanks also to the crew at Caltech who provided the transcription of the talk and to Jim Melancon at ISI for editing the transcript into a reasonable form for publication.

# FAST TURNAROUND FABRICATION FOR CUSTOM VLSI

Gunnar A. Wetlesen  
VLSI Technology, Inc., Santa Clara, California

Now, you are probably wondering how somebody on crutches is going to be able to provide fast turnaround or anything. Well, excuse me while I grapple with everything that I have to carry up here.

While we are in an informal mode, I would like to say that we made an acronym of an acronym, and we go by VTI, standing for VLSI Technology Inc.

The so-called foundry concept as first proposed by Carver Mead is one of the areas of business that VTI hopes to support. I am not sure if our coming about was to make Carver's prognostications accurate, or whether it is indeed a reflection of the support that we have gotten from a number of investors. I would like to acknowledge them. There is a computer company with vision and venture capitalists that are represented in the audience today that made VTI come about. Previously, the idea of the foundry was only available internally in large corporations, such as Hewlett-Packard, who did the prototyping work for the MPC runs that Lynn Conway described earlier. We are going to be one of the factors in that area in the future.

So, beginning with the more formal portion of this talk, I will be addressing the whole area of custom VLSI, and addressing the MPC area just a little bit.

The use of custom and semi-custom circuits is rapidly growing in many electronic equipment areas, particularly as the potential of VLSI is being realized. Helping create this intensity in custom activity are the following advantages available through customization seen in contrast to the use of standard products:

- 1) A competitive advantage -- product differentiation and/or proprietary feature sets.
- 2) A performance advantage -- reduced component count and associated system overhead.

Realizing the potential of custom VLSI circuits is analogous to the emergence of the microprocessor a decade ago. In this case it is presently limited by the front-end design phase (both schedule and cost). Thanks to many of the people present here today, the design mechanism for translating systems into silicon is evolving rapidly.

Regardless of the design mechanism or the driving force toward custom in terms of systems advantages, all custom circuits must subsequently navigate the same obstacle in development: the time-consuming and often iterative prototyping phase.

Fast turnaround for custom VLSI fabrication will now be examined. This fabrication need will be translated into terms of the characteristics desired of the MOS/IC manufacturer. We believe there is a need in the semiconductor industry for new kinds of companies properly postured to service the requirements for the coming era of custom VLSI.

Fast turnaround for custom VLSI fabrication as a goal defines three general requirements which must be supported for the goal to be met. They are:

- 1) VLSI level of fabrication technology.
- 2) Supporting custom designed circuits (as opposed to standard products).
- 3) Providing fast turnaround fabrication.

Let us consider now each of these in detail and the resulting "factory" characteristics in terms of operating philosophy, organization, equipment, and people.

#### VLSI FABRICATION TECHNOLOGY

There are several common definitions of what is VLSI. While best defined in terms of system complexity, other definitions are often based on feature size or device complexity. For example, the state-of-the-art 64K dynamic memory has been referred to as a VLSI chip by virtue of its greater than 100,000 transistor count (figure 1). Certainly, the domain of logic circuits above  $10^5$  transistors can be associated with VLSI system complexity.

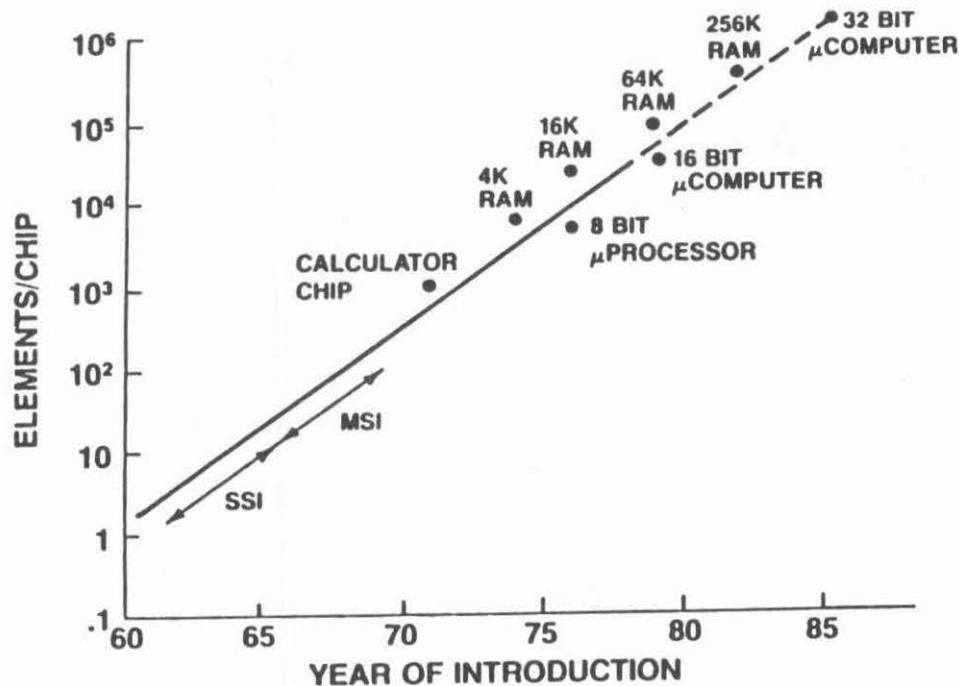


Figure 1

What kind of fabrication technology for VLSI? as a practical matter, fabrication processes will evolve with time. Current technologies with VLSI potential included scaled n-MOS (commonly called H-MOS) and oxide isolated CMOS. One thing that is clear is that process technologies are becoming more

specialized depending on their application (figure 2). For example, in memory technology, double poly and diffused capacitors have been evolved to optimize the density of dynamic RAMs whereas static RAMs have incorporated high value polysilicon transistors and multiple transistor thresholds for both density and performance. I think it can be concluded that evolution of technology for VLSI custom circuits will be on a different branch of the technology tree compared to memory circuits. In particular, the use of multilevel metal as a solution to the interconnect and signal delay problem is overdue for logic circuits. Additional benefits would be gained by features improving ROM and PLA densities.

---

### **SPECIALIZATION OF PROCESS TECHNOLOGY**

---

<u>PRODUCT TYPE</u>	<u>SPECIAL PROCESS FEATURES</u>
DYNAMIC RAMS	<ul style="list-style-type: none"><li>• DOUBLE POLY</li><li>• ENHANCED CAPACITORS</li></ul>
STATIC RAMS	<ul style="list-style-type: none"><li>• DOUBLE POLY (INTERCONNECT)</li><li>• POLY LOAD RESISTORS</li></ul>
ROMS	<ul style="list-style-type: none"><li>• SELF-ALIGNED CONTACTS</li><li>• MULTIPLE THRESHOLDS</li></ul>
LOGIC CIRCUITS	<ul style="list-style-type: none"><li>• MULTILEVEL METAL</li></ul>

Figure 2

In terms of commercial state-of-the-art equipment, the current noncap-tive lines are still based on 1:1 projection aligners. Selective use of direct wafer steppers on certain critical mask levels is being employed for manufacture of volume standard products. Due to the logistics and the potential for repeating defects associated with changing 10x reticles, the use of 1:1 projection printers is favored for producing lower volume custom and customizable circuits. With use of planar plasma etching and reactive ion technologies, resulting average feature sizes of 2-3 microns appear readily manufacturable with projection aligners in the near future (figure 3). Other aspects of equipment selection will be considered later.



## MOS COMPLEXITY/DIMENSION FORECAST

YEAR	DEVICES PER CHIP	DIE SIZE (mils)	AREA (mils <sup>2</sup> ) PER DEVICE	LINE WIDTH
1978	35,000	200 x 200	1.26	4 $\mu$
1982	250,000	300 x 300	.36	2.1 $\mu$
1985	1,000,000	400 x 400	.16	1.4 $\mu$

Figure 3

### SUPPORTING CUSTOM CIRCUITS

Supporting custom circuit manufacture is markedly different than the merchant IC industry supplying standard products. The first and most obvious characteristic is that a custom circuit is by nature unique. This uniqueness begins with the generation of the mask set, and involves both a significant data transfer of the base description of the circuit layout, and also the practical details of mask geometry polarity, and skewing, and process monitor chip insertion. The industry needs to develop clean, automated procedures for easy customer interfacing.

The first step is to replace the personal phone call and the physical transfer of data base tapes, which will be accomplished instead by what we call "VTI Net." Access to VTI via a commercial computer network allows the transfer of design files in CIF and other formats (figure 4). More importantly, an inquiry service will be established to answer the basic questions a designer in a remote location needs to ask in order to interface with our fabrication service, including cost and scheduling information as well as formats, etc.

---

**NETWORK ACCESS TO VTI ("VTI NET")**

---

- VIA COMMERCIAL NETWORK
- ACCEPTS DESIGNS IN CIF (AND OTHER FORMATS)
- ALLOWS FOR BASIC INQUIRY RESPONSE SERVICE

Figure 4

Central to such an arms-length service working are standardized interfaces before and after fabrication (figure 5). In particular, the technology and design rules must be common. nMOS based on lambda rules will be offered

---

**STANDARDIZED PRE-FABRICATION INTERFACE**

---

- BASED ON LAMBDA DESIGN RULES
- REQUIRES E-BEAM MASK GENERATION
- "STARTING FRAME" INSERTED BY VTI

Figure 5

plus specials on a "non-network" basis (figure 6). Mask generation will be done using an electron beam system not only for dimensional and complexity reasons but also because it greatly simplifies the starting frame task. Alignment keys and critical dimension measurement points need not be inserted on the circuit as those are included within the process control monitor (PCM), chip inserted by VTI in the mask set.

---

**CUSTOM INTERFACES**

---

**TOOLING**

- DATA BASE
- MASKS

**FABRICATION**

- VERIFICATION
- SPECIAL PROCESSING

**TESTING**

- WAFER LEVEL
- PACKAGE LEVEL

Figure 6

This same PCM serves as the common denominator for evaluation of the processed wafer lot (figure 7). It provides process parametric characterization data automated in both measurement and data reduction. VTI intends to

---

### **STANDARDIZED POST-FABRICATION INTERFACE**

---

- **PARAMETRIC CHARACTERIZATION VIA PROCESS CONTROL MONITOR (PCM)**
- **RESULTS OF "CANARY" CIRCUIT EVALUATION (OPTIONAL YIELD DATA)**
- **PROTOTYPE PACKAGING (FOR CHIPS MEETING VTI ASSEMBLY STANDARDS)**
- **ARCHIVAL OF PCMs (FOR OPTIONAL RELIABILITY AND OTHER EVALUATIONS)**

Figure 7

take advantage of NBS work and to make this PCM and support documentation widely available to provide a standard suitable for multiple sourcing of fabrication services. The PCM will provide, however, only limited information in terms of yield analysis. In addition, VTI will include a "canary" circuit, for example a large shift register, whose functionality will be part of the wafer acceptance criteria. In early prototyping phases statistically significant quantities of these test circuits can be incorporated into the mask set for correlation and projection of both yield and circuit performance. The PCMs will be saved in die form when a packaged chip rather than wafer level interface is employed, so that they can be used both for later electrical characterization and for quality and reliability verification for military programs. Similarly, correlation with performance of both the prototype circuit and the "canary" will be even more valuable if process variants or "tweaks" are employed either to give special features (such as on-chip analog interfaces) or to give performance enhancements.

In addition, the "VTI Net" will also be set up with multiproject chip (MPC) capability in mind, similar to that described earlier by Lynn Conway (figure 8). Initially, the MPC capability will be used internally to support VTI's design courses. Based on a core of material available on video tape, these courses are being given in remote locations for appropriately equipped companies which will in turn enjoy MPC availability.

---

**FUTURE MPC CAPABILITY PLANNED**

---

- INITIALLY INTERNAL FOR VTI COURSES
- INTERIM AVAILABILITY ON A SELECTED CUSTOMER BASIS
- NETWORK CAPABILITY IN FUTURE (LOGISTICAL, TECHNICAL AND PROPRIETARY ISSUES)

Figure 8

In the future, the courses will be given at VTI's planned design center. When technical and proprietary issues are solved, the MPC capability will be expanded and will become available for routine prototyping on an individual basis via the network.

Let us now consider the workings of the factory which supports the fabrication of custom VLSI. As a result of the percentage of prototype runs and the larger number of unique mask sets being run to get suitable volumes in a given manufacturing module, the support organizations of a custom circuit factory must be substantially different than current merchant IC suppliers. The greater logistics task begin with order entry, where a technical transfer is implicit, and continues through the factory with production control, quality assurance, and other organizations being matched accordingly. Clearly, the task of performing on a build-to-suit basis at each step from maskmaking through fabrication, test, and assembly, while maintaining a line item orientation, is much greater than that for the inventory orientation used in standard products. The operating philosophy completely different, perhaps best described as the need to be effective rather than efficient, especially in the context of fast turnaround. The caliber and awareness of people in these support organizations must be matched accordingly. The mainstay of their equipment must be real time control systems for scheduling and tracking programs throughout the factory cycle. Ultimately, status information should become available as part of the inquiry service over the network.

**PROVIDING FAST TURNAROUND**

In traditional IC companies, factory production cycles of 16 weeks, assuming mask availability, are not uncommon. Of this cycle typically 6-8 weeks is in wafer fabrication. Clearly, this cycle is untenable for circuit prototyping -- a year could be consumed in 2-3 iterations. As will be seen, however, carryover of a fast turnaround philosophy into production also brings tangible benefits.

The theoretical limit on fabrication cycle time of a typical MOS process is on the order of 2-3 days. A practical goal for minimum lot size fabrication of prototypes is one week if the factory is organized accordingly to minimize waiting in queues before each operation. The preeminent requirement in achieving fast turnaround is the operating philosophy (figure 9). Instilling in the work force the concept that "minutes count" is a further

<b>COMPANY CHARACTERISTICS</b>		
<u>COMPARISON</u>	<u>EXISTING IC CO's</u>	<u>PROPOSED CO</u>
GENERAL ORGANIZATION	BASED ON EFFICIENCY	BASED ON EFFECTIVENESS, PERFORMANCE TO SCHEDULE
PHILOSOPHY	INFLEXIBLE  COST BEFORE SERVICE PRODUCT ORIENTATION INVENTORIES COUNT	FLEXIBLE (PROCESS, PRODUCT, ETC.) SERVICE ORIENTATION CUSTOMER ORIENTATION TIME COUNTS

Figure 9

development of this theme. In addition to the scheduling and tracking organizations operating as a "real time" function, the other organizational strategy for providing fast turnaround of prototypes is the pilot line concept (figure 10). Located within the manufacturing facilities to assure ease and success of later production transfer, a prototype pilot line can, by providing the necessary focus and priority, reduce fabrication cycle times severalfold. In addition to providing an elite team of skilled people, it is necessary to give last-in, first-out priority on shared equipment plus provide dedicated equipment at steps where lot uniqueness is maintained, such as photomasking. Furthermore, factory equipment must be more redundant to provide capacity for "surge conditions" and be chosen to be not too "state-of-the-art" so as to maintain high up-time.

---

**CYCLE TIME COMPARISON**


---

**I PROTOTYPES**

<u>STEP</u>	<u>TYPICAL</u>	<u>POSSIBLE</u>
MASK MAKING	4-6 WEEKS	3 DAYS
WAFER FAB	4-6 WEEKS	9 DAYS
PACKAGING	2-3 WEEKS	3 DAYS
TOTAL	10-15 WEEKS	3 WEEKS

**II PRODUCTION (WITH EXISTING MASKS)**

<u>TYPICAL</u>	<u>POSSIBLE</u>
12-16 WEEKS	6-8 WEEKS

Figure 10

Wafer fabrication is only one link in the turnaround chain. Mask making and assembly are others. In-house control of an e-beam mask generation system can keep this cycle time minimized. Typical writing and processing times make one plate per hour realizable with additional time for inspection. As a result, two- to three-day turnaround for a mask set is practical while still providing priority for single layer redos or customized devices. Similarly, an in-house prototype packaging line can assemble prototypes in a couple of days.

Summing up these individual times it can be seen that less than a week total cycle time from receipt of tapes to custom prototypes could be routine if an IC manufacturing facility were postured as described. Moreover, the production cycle times could easily be halved compared to traditional IC suppliers in this same atmosphere. In terms of limiting exposure to upstream yield or reliability problems as well as responding to increased customer needs, shortening of the manufacturing pipeline is an equally attractive possibility.

**CONCLUSION**

VTI has been funded to serve the need of a high technology for VLSI and a service-orientation to provide quick turnaround.





## LONGER TERM DIRECTIONS FOR SEMI-CUSTOM VLSI

Gordon B. Hoffman  
United Technologies Microelectronics Center,  
Colorado Springs, Colorado

Through the convenience of jet travel I have been able to talk both in Detroit and here in Pasadena today--although I admit the time zone change helped, too. As I was cruising along I realized that next year we will begin limited production of a set of semicustom CMOS gate array chips for the jet engine fuel control on the Pratt & Whitney engine that has recently received so much attention due to large orders from Delta and American Airlines. A year or two after this production begins, my life, as well as yours, might well depend on the performance of that controller. It's a sobering thought that tends to bring home the reality of the technologies we develop.

Our newly formed company, United Technologies Microelectronics Center, or UTMC for short, is dedicated to the development and design automation of semicustom circuits. Before UTMC was formed last year, all of the divisions of United Technologies had to go outside for their custom IC needs. It was getting increasingly difficult to get support from the merchant semiconductor industry, particularly where low volumes of devices were required.

In 1979 United Technologies acquired Mostek, and there was hope that Mostek would help alleviate this difficulty in obtaining custom integrated circuit support. Now those who know Mostek realize that "custom" is not a happy word at Mostek. "Custom" represents to Mostek the wrong use of scarce design resources. Mostek does not disagree that there is a strong market need, but for Mostek it has been a conscious business decision not to participate. Mostek did not originally hold this view; in fact, the first one-chip calculator circuit was made by Mostek on a custom contract. The original customer is now bankrupt, however, which gives you an inkling of why Mostek is leery of the custom business.

The custom needs of the UTC divisions range from five- to ten-chip sets a month for esoteric military applications to high-volume chips for automotive applications. Because of the mismatch with Mostek, a high-volume MOS commodity IC supplier, a joint study was launched by UTC and Mostek to

find a solution. As a result of that study UTMC was created last year in Colorado Springs and funded with \$22M for Phase 1. Our Phase 1 goal is the development of a CMOS gate array design system by the end of 1982. Mostek will benefit as well as the other UTC divisions, both as a user in its system divisions and in its ability to offer semicustom services to the merchant market using the tools we develop at UTMC.

Gate arrays were chosen as our initial thrust because automation of that design style is possible in the time scale we had to work with, that is, ASAP. Gate array design turnaround time is rapid, and production costs are reasonable, even for the automotive volumes some of our divisions require.

Since our goal from the outset was an automated design system, actual design of the arrays and processing considerations are secondary to CAD requirements. For example, if we can simplify our software or make it easier to use by slightly increasing the wafer processing cost, then we'll live with the extra processing cost. In fact, one of our first decisions was to use two-level metal with our CMOS arrays to make the routing problem easier and the corresponding software task smaller.

Preoccupation with chip area is another concern that needs examination with semicustom circuits. Yields for gate arrays and other semicustom approaches tend to be much higher than custom circuits of corresponding chip size. Several factors are involved including a lower active density, fewer design rule violations particularly with automated design systems, and the fact that a cumulative learning curve generally applies to semicustom circuit cost independent of the actual customization. It's really the number of good die per wafer that determines chip costs, which is in turn determined by a combination of chip size and yield. A gate array twice as large as an equivalent custom chip, but with twice the yield, has the same chip cost, and as it turns out, chip cost becomes less important as system functions require more expensive, high-pinout packages, and as other system integration cost savings are taken into account. I'll talk more about this subject a little later.

We will have the capability to merge several different array metallizations onto the same mask set and, therefore, on the same wafer. This will

allow us to economically process very small quantities of devices, even breadboard quantities.

The complete cycle from the start of logic design to delivery of custom chips should require 3 to 6 months vs 18 months or more for traditional custom circuits. Logic design for the jet engine fuel control I mentioned earlier began last summer and the first complete chip set will be delivered this February.

Despite manual placement and routing of these four chips, each with about 6000 CMOS pairs, only seven months were required for the entire design cycle, so I feel very confident about the 3 to 6 month projection using design automation.

This overall system really represents a type of foundry operation, connecting silicon to order. The concept can, of course, be extended to other forms of semicustom IC's and I'll talk about that later.

Now many of you realize that semicustom concepts, and gate arrays in particular, have been talked about for more than 10 years. Systems similar, at least in concept, to the one I've just described have been tried but without commercial success, that is, until recently. IBM has used gate arrays quite successfully, as has DEC, AMDAHL, and Storage Technology, to name a few. I feel it's instructive to see why semicustom approaches are experiencing a "renaissance" after a long period of "dark ages," which should give an insight into future directions. Let me share with you some of the thoughts along this line that lead to our entry into the semicustom arena.

The first observation was that of transparency. If we can make silicon design transparent to logic designers then we have solved the shortage of silicon designers that so limits the interest of merchant semiconductor suppliers in custom design. Our design automation system is an attempt to do just that. Not only is the initial design time reduced, but so is the time for the inevitable design changes that always seem to occur in custom designs.

When simulation, testability verification, and array routing are complete, the divisions will send us, via DECNET, data that contains test patterns, routing, and identification. We will derive the actual test

equipment programs from these data and forward mask-making and identification information on to Mostek. It is interesting to note that our customers need not divulge the chip's function, and it would take quite an effort to derive its function from the data they supply us.

At Mostek three interconnection masks will be generated: first metal, second metal, and the vias between the two. E-beam mask-making equipment will be used, although we envision going to E-beam direct write on wafer by 1983. Preprocessed CMOS gate array wafers will be inventoried at Mostek with the first level of aluminum already applied to the wafer. We expect the turnaround time for mask-making and the completion of metalization to take two weeks or less, rather than the 13 to 18 weeks normally required for a complete set of CMOS masks and wafer processing from bare silicon.

When Mostek is finished with the metalization, they will deliver the customized wafer to us and we will probe, assemble, final test, and environmentally process the finished devices prior to shipment. Our initial packaging efforts will utilize leadless chip carriers, or a leaded version of these.

The next reason behind the semicustom renaissance is performance. The metal-gate PMOS arrays of the early 70's had a narrow application, since they were not fast enough to replace TTL in most digital systems. Today, with bipolar or CMOS arrays, gate delay times can be achieved such that TTL replacement, including Schottky TTL, is quite practical, greatly expanding the size of the potential market. With CMOS we anticipate average on-chip gate delays under 3 nsec using double-level metal and 3.5 micron gates, and there is still a lot of room for further improvement.

There is also ability to insert technology improvements without modifying the basic functional design. We expect, for example, to be able to process the same array metalization on differently processed arrays for radiation hardened applications or very high-temperature needs.

The last and perhaps most important attraction to semicustom circuits is economics. While traditional logic design costs and associated breadboarding, documentation, and preparation for manufacturing costs continue to increase, computer time costs have dropped dramatically. Some of you remember the discretionary-wired LSI program that was much touted in the

late 1960's. At that time I recall the computer run time costs alone to map and route interconnections to the good die on a two-inch wafer amounted to \$2,000. Remember those are 1968 dollars, or about \$4,700 in today's rhubarb currency. Based on a recent study, using an IBM benchmark program, computer costs have declined so much that \$2,000 spent in 1968 would only cost \$40 today, or \$17 in 1968 dollars, and computer run costs are continuing to decline. In fact, computer costs may be dropping as fast as MOS RAM prices are, and you can't say that for many items in today's world.

The economic benefits of system integration onto silicon are well known and have been the driving force toward higher levels of integration. But access to higher levels of integration has been limited to high-volume system manufacturers that could justify traditional custom design costs, or to smaller users through microprocessors and related standard products. The immense volume of TTL and other forms of small and medium scale integrated circuits demonstrates that the transition to higher levels of integration is far from complete, and is an attractive area for penetration by semicustom circuitry.

Inflation has exacerbated the problem not only by increasing the cost of the small and medium scale integrated devices themselves, but the cost of putting them into systems and maintaining operation of those systems has risen dramatically. A \$100 124-pin semicustom gate array, for example, may be a real bargain if it replaces 60 TTL packages, eliminates a PC board and edge connector, and saves on service and repair costs. Other system advantages include the potential elimination of cooling fans, smaller and less costly power supplies, lower system manufacturing cost, smaller cabinet volume requirements and therefore more cabinet styling latitude, and I could go on to list at least 10 more advantages of system integration but I'm sure you could come up with interesting lists of your own.

Finally, under the topic of economics are "market window costs" to a systems manufacturer. There is a measurable dollar value to getting a product into the market early, or at least before competition gets there. As experience shows again and again, the highest return on investment belongs to the manufacturer who dominates market share, and it's hard to get this domination if you are a year or two late with your product introduction. With semicustom circuit design, times are short, production ramps up quickly

and, I believe, without the number of unexpected design fixes often required with traditional custom circuits, or conventional designs using TTL for that matter, and necessary changes are quickly implemented.

Market window costs may be a big factor in keeping even high volume manufacturers from initially going the traditional custom route, and I wonder if the switch to custom would occur even if economics were favorable. At the design automation conference last year I asked a European manufacturer of gate arrays, who has fabricated over a thousand different customizations, how many were eventually designed out and replaced by smaller custom circuits. His answer was none, to his best recollection. It seems the engineering and financial resources to make the conversion were always put into new product designs, where the return on investment was consistently higher. I expect this is not an isolated phenomenon.

Now I've talked about what we are doing with gate arrays, why the semicustom approach is experiencing a revival, and the critical role design automation will play. My perspective, admittedly, has been from the MOS point of view, although there is quite an activity in bipolar technology as you know. I now think it's time to make some predictions beyond where my earlier comments leave off.

I don't see gate arrays being a temporary phenomenon. The quick customization will be critical in my applications, despite chip size implications, at least until automated fab operations can produce chips from bare silicon in a really short time frame. There's a good deal of architectural innovations on the horizon for gate arrays as well. We have a big problem to solve with testability, but that problem is endemic with all efforts toward higher levels of integration, and must and will be solved.

The next logical step is to use standard cells, where the design system is identical to gate arrays, except that the wafers are processed from bare silicon using compact, although dimensionally constrained, layouts for each cell type, and resulting in a smaller chip. This approach would be a cost-effective transition from a gate array design for high volume applications once the system design has been stabilized. We intend to make standard cells a key part of our second-phase effort, which begins in 1982.

Following the standard cells I can see a "macro cell" approach where macro cells could be subsystems themselves, not constrained to particular shapes or positions. In the cell library there might be a 32-bit processor, speech synthesizer elements, memory, standard cell logic blocks, and so forth. The design automation requirements are hardly trivial, necessitating, for example, some sort of high-level machine description language. After all, we can't keep dealing with simple logic design inputs as the level of integration continues to rise.

And I couldn't finish this talk without a comment on foundries. It's clear to me that with design automation, connecting silicon to order will become a way of life. The order initiators will, in time, be predominately the custom community rather than the semiconductor manufacturers themselves. To make this practical, design automation is the key. Semiconductor wafer fab and backend operations must evolve from an orientation toward making huge volumes of relatively few circuit types to small volumes of many circuit types. Our plans are to develop the systems I have described today and work with Mostek to define and develop such a foundry operation in the mid 1980's.

Let me leave you with one last thought by proposing a definition of semicustom circuits, and then leaving you with a curious observation.

Semicustom integrated circuits: those ICs which appear custom to the user but are standard to the manufacturer. If this definition is a fair one, and I believe it is, then the distinction between semicustom and custom is bound to disappear. Design automation at the device level is progressing rapidly and transparent custom circuit design can't be far off. When it does become a reality then the only distinction might be entry level, device versus logic, or some other level, but I suspect that will merge as well. I will leave the conclusions to you.





*FABRICATION SESSION*

*Chairperson: James D. Meindl  
Professor of Electrical Engineering  
Stanford University*

## FABRICATION SESSION

VLSI is governed by a hierarchy of limits. Each of the five levels of this hierarchy is constrained by all preceding levels. Briefly, limits representing each level are

- (1) A fundamental limit derived from thermodynamics requiring the energy expenditure per switching transition in a digital system to be  $E_s \geq 4kT = 1.65 \times 10^{-20}$  joules.
- (2) A material limit for silicon requiring the transit time of an electron ( $\Delta t$ ) through a potential drop  $\Delta V$  to be  $\Delta t \geq \Delta V / v_s \epsilon_c = 0.416$  ps for  $\Delta V = 1V$ ,  $v_s$  the scattering limited velocity and  $\epsilon_c$  the critical field.
- (3) A device limit for an N-channel MOS transistor to avoid punch-through requiring a source-to-drain spacing  $L \geq 2[2\epsilon_{Si}(V_D + \phi_{bi})/qN_A]^{1/2} = 0.18\mu m$  for substrate doping  $N_A = 1.5 \times 10^{17}/cm^3$  and drain voltage  $V_D = 0.5$  V.
- (4) A circuit limit for a CMOS inverter requiring an average power drain  $P_{AVE} = C_L V_{dd}^2 f$  where  $C_L$  is the load capacitance,  $V_{dd}$  the drain supply voltage and  $f$  the frequency of excitation.
- (5) A system limit. This level consists of a number of sublevels which include both software and applications constraints. Consequently, system limits represent the most numerous and nebulous group of the hierarchy albeit potentially the most profoundly important.

Any level of the hierarchy can be split into two generic metalevels. At the circuit level the conceptual metalevel is intrinsic or independent of the fabrication equipment of technology used to produce a chip, while the practical metalevel depends, for example, on whether photolithography or X-ray lithography is used.

The papers in this session of "fabrication" deal largely with practical limits in VLSI at the device and circuit levels.

## TRENDS IN SILICON PROCESSING

V. Leo Rideout  
IBM Corporate Headquarters  
Armonk, NY 10504

Abstract

The advent of very large scale integration will require substantial progress in all aspects of silicon technology: processing, lithography, modeling, design tools, chip architecture, and applications. This paper will survey current trends in silicon integrated circuit fabrication, focusing on new developments and outstanding problems. Progress in both bipolar and MOSFET technologies will be considered. Silicon fabrication techniques will be described in terms of the repetitious application of operations that are additive (oxidation, doping, deposition), selective (lithography), and subtractive (etching). The objective of these operations is a reliable and predictable device structure. Device structures will be described in terms of isolation areas, devices, contacts (intraconnection vias), wiring (interconnection lines), and passivation. Immediate problems in isolation size, device performance, contact resistance, and wiring topography will be identified. Future needs for improved structures will be indicated. Promising new approaches such as lightly-doped drain FETs and silicide-on-polysilicon (polycide) wiring will be described. Throughout this discussion the importance of process modeling will be emphasized.

## 1.0 INTRODUCTION

The advent of very large scale integrated circuits (ICs) will require substantial progress in all aspects of silicon technology: processing, lithography, modeling, design tools, chip architecture and applications. This paper will survey current trends in silicon integrated circuit fabrication including processing techniques, lithographic tools, and process modeling. As illustrated in Figure 1, the level of integration in mass manufactured IC chips has reached 160,000 components on a 64 Kbit dynamic random-access memory (RAM) chip, 80,000 on a 16 Kbit static RAM, and 40,000 on a 16 bit custom microprocessor (1). By the end of this decade we can expect 512 Kbit dynamic RAM's and 128 Kbit static RAM's with one million components per chip, and 64 bit microprocessors with 250,000 components. By the term component we mean an integrated transistor, resistor, capacitor, or diode.

In 1979, the worldwide sale of semiconductors was \$11.1 billion, most of it in silicon IC's. In 1980, semiconductor manufacturers are expected to invest well over \$2.0 billion in new fabrication plant and equipment. Including captive silicon suppliers, the total investment will be in excess of \$3.0 billion. Obviously silicon processing is a big business. Presently, the most advanced bipolar and MOSFET circuits are mass produced with photolithographic groundrules of 2.5 to 3.0 microns. This is expected to improve to 2.0 microns in 1982, and to 1.0 microns by about 1988 (see Figure 2). New processing tools and techniques needed to maintain this progress and novel device structures evolving at smaller dimensions will be discussed in this paper.

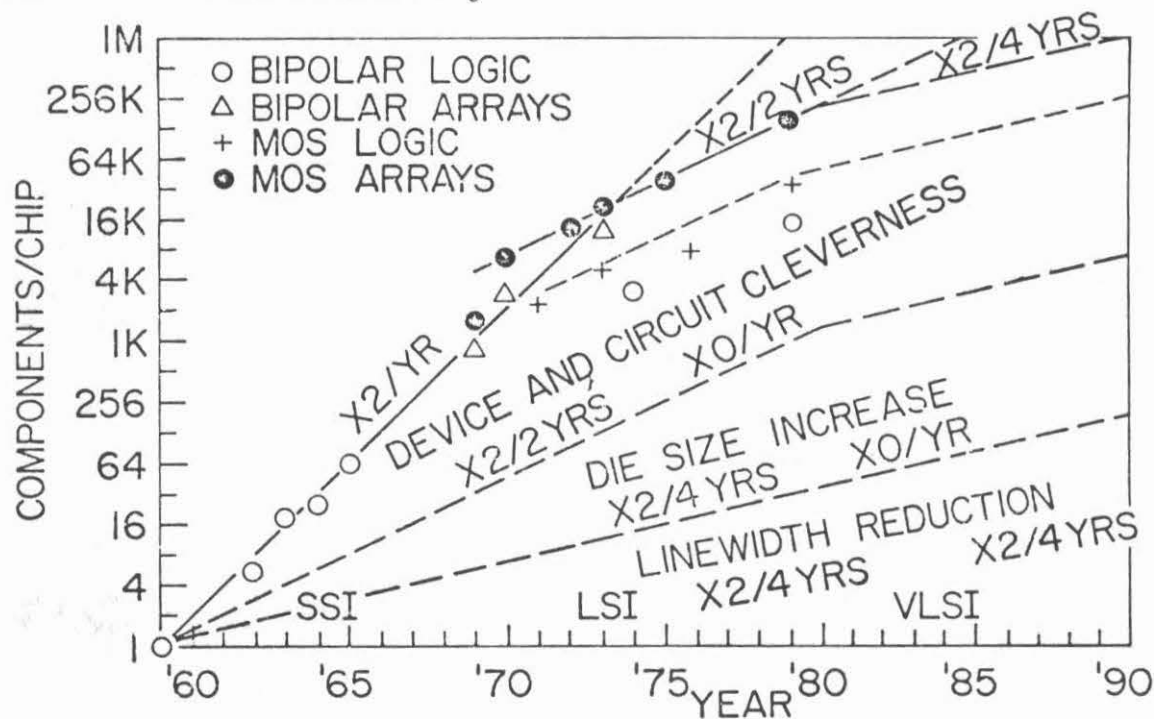


FIGURE 1: Yearly rate of improvement in components per chip.

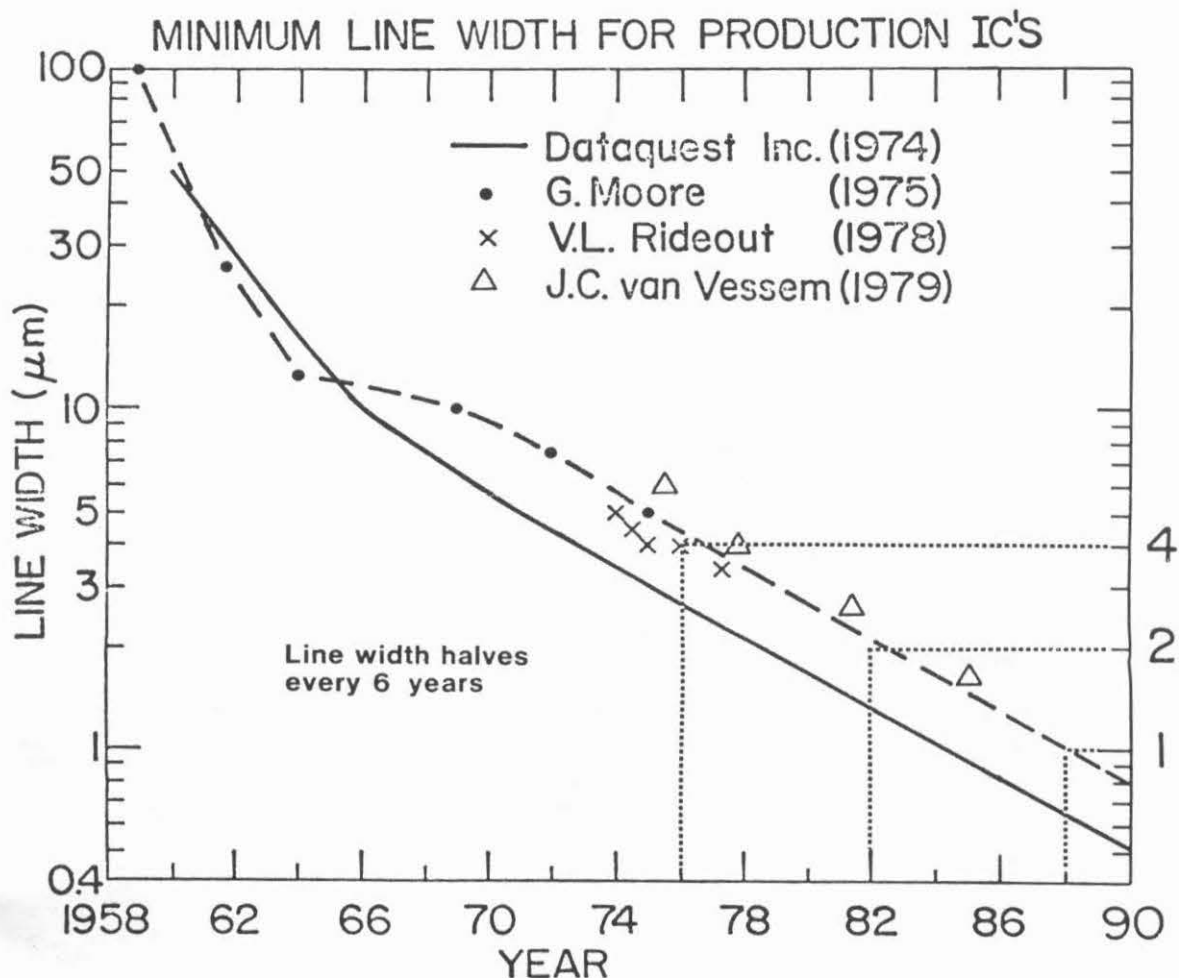


FIGURE 2: Progress in lithographic linewidth.

CALTECH CONFERENCE ON VLSI, January 1981

## 2.0 ADDITIVE OPERATIONS

Silicon IC fabrication may be thought of as the repetitious application of materials processing techniques that are additive, selective, and subtractive in nature. Typically, only one lithographic masking operating or step is associated with each loop of this repetitious manufacturing procedure. It will be assumed that the reader is at least basically informed about the fabrication process for silicon integrated circuits (2). The ability to thermally grow a layer of silicon dioxide either locally or globally on a silicon wafer is one of that semiconductor's most important properties. The oxides are usually grown in dry oxygen (thin layers/slow growth) or in the presence of water vapor (thick layers/fast growth). Chlorine ions, usually in the form of HCl vapor, can be added to reduce oxide charge and improve capacitor characteristics, particularly in MOSFETs. Oxidations are typically performed in the 900-1100°C range for times of 10 to 100 minutes.

A major direction in oxidation techniques is toward lower temperatures (less than 1,000°C) and higher pressures (more than one atmosphere). The desirability of lower fabrication temperature is quite general because the drive for continually shrinking lateral dimensions (i.e., device scaling) has led to thinner vertical layers and reduced processing times. The processing time reduction can become so severe (only a few minutes) that uniformity control is impaired. In order to maintain or even lengthen processing times for better control, lower processing temperatures are desired. Other advantages of lower processing temperature include reduction of out-diffusion, grain growth, defect generation, and wafer warpage.



## 2.1 OXIDATION

During oxidation, both the linear and parabolic rate constants increase with the partial pressure of the gaseous oxidant (3) (see Figure 3). This advantageously leads to a much faster oxidation rate at a given temperature or the same oxidation rate at a much lower temperature. For every increase of one atmosphere, the oxidation rate doubles. Alternately, if a fixed rate is desired, every increase of one atmosphere allows a reduction of 30°C in oxidation temperature. Potential advantages include reduced thermal-induced damage (e.g., wafer warpage and oxidation-induced stacking faults), lower surface-state density, and reduced boron depletion. Presently, pressurized furnaces are commercially available for dry and wet oxidations at pressures up to 25 atmospheres. This technique can be expected to be incorporated into production in the near future.

## 2.2 CHEMICAL VAPOR DEPOSITION

Thin layers can also be chemically vapor deposited (CVD) from gaseous sources in an RF induction-heated furnace to provide epitaxial silicon (900-1300°C), polysilicon/SiO<sub>2</sub>/Si<sub>3</sub>N<sub>4</sub> (600-1000°C), and passivation (600°C) layers (4). Dopants can be incorporated into the chemical deposition process. A major trend is to use low pressure CVD (0.1 to 40 Torr) which affords improved thickness control (especially for polysilicon layers), reduced auto-doping, and higher throughput. The deposition temperatures listed above could potentially be reduced by 100 to 200°C. Probably the most difficult deposition process to improve on is single crystal silicon-on-silicon epitaxy which is essential to bipolar processing. Two interesting research techniques that address this problem are molecular beam epitaxy or MBE (5) and solid phase epitaxy or SPE (6) which utilize substrate temperatures of 400 to 600°C.

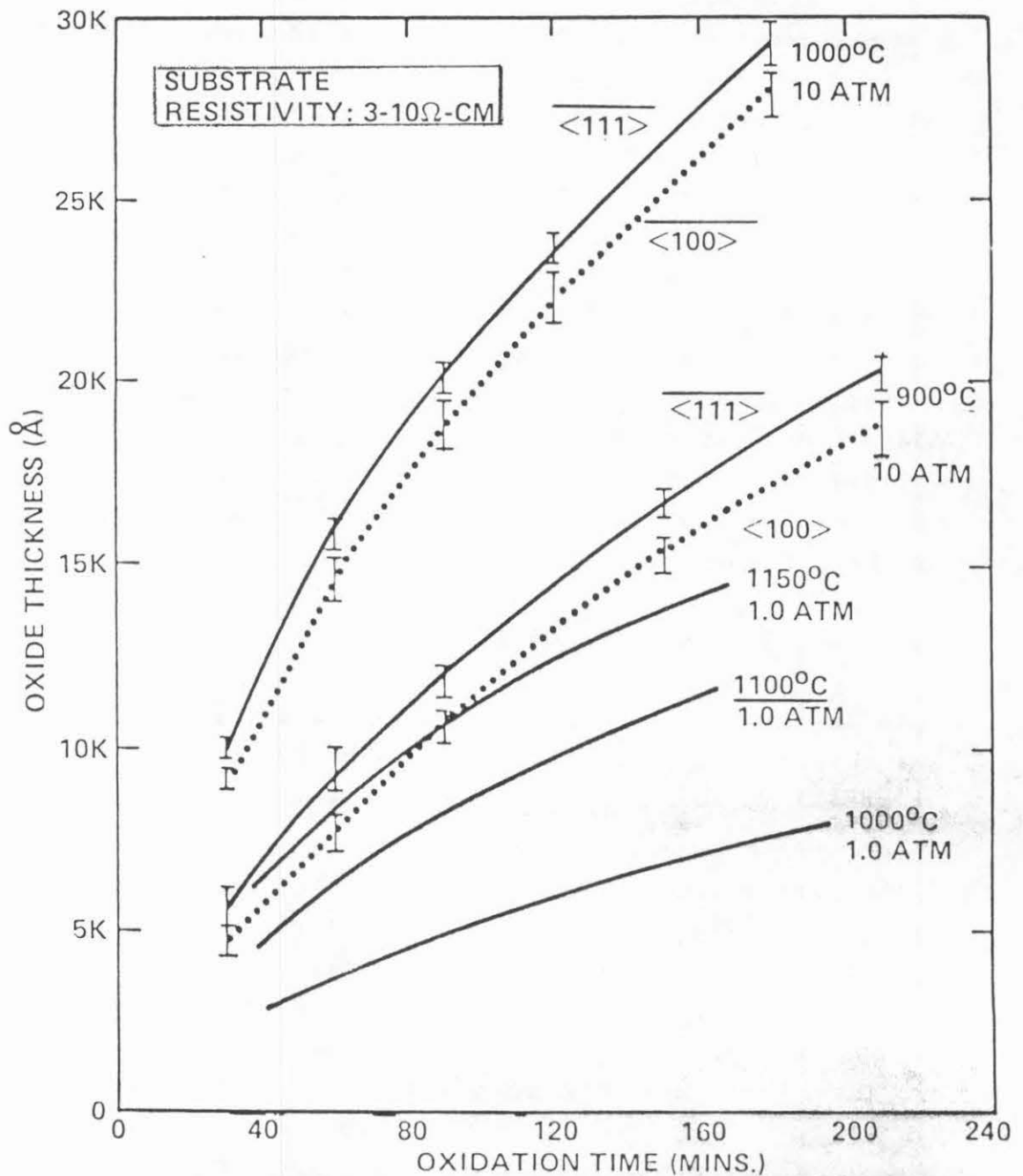


FIGURE 3: Wet oxidation growth curves for one and ten atmospheres (after Ref. 3).

The chemical vapor deposition rate can be enhanced by presence of an RF plasma. The gaseous reactants (e.g., nitrogen, ammonia, and silane) interact to form a solid film product and other gaseous by-products (7). The reaction is sustained by the RF plasma rather than by external hot-wall heating. Low deposition temperatures (200 to 400°C) and highly conformal films are the result. Plasma deposition of silicon nitride is now widely used for final passivation as a replacement for phosphorus-doped silicon dioxide. A related future activity is photo-excited CVD of silicon nitride, silicon dioxide and, possibly, epitaxial silicon.

Molecular beam epitaxy (5) utilizes a vaporized beam in ultra high vacuum. The MBE technique has exhibited excellent thin film quality (one micrometer thickness) but is hindered by throughput limitations and equipment cost. With solid phase epitaxy, a doped amorphous silicon layer is deposited onto the substrate, and an epitaxial film is produced by heating the composite either locally or in a furnace. N-channel MOSFETs have been fabricated in SPE-grown epitaxial layers with channel mobilities of 360 to 480 cm<sup>2</sup>/V-sec (6). In addition to low temperature growth for improved layer thickness control and reduced dopant redistribution, SPE offers selected-area epitaxy which can be attractive for defining isolation regions. This can be achieved by depositing onto a masked substrate, or by local heating with a laser or electron beam.

### 2.3 BEAM HEATING

In several processing operations it is necessary to subsequently heat or anneal the wafer. One such example is the annealing of regions doped by ion implantation to remove local stress and to activate the dopant. Traditionally a RF heated furnace with an

inert gas ambient is used for annealing in the 800-1000°C range for about 30 minutes. Significant dopant redistribution can occur during the annealing step, but laser or electron beam heating offers the potential for fast local heating that avoids this. Obviously, the beams can be scanned to anneal the entire wafer. Potential advantages are reduced processing time and lower cost. Thus far beam heating techniques are not reliable enough to anneal active semiconductor regions for production devices. The future applications of beam annealing in order of acceptance probably are:

- inducing backside damage gettering,
- forming silicide layers,
- activating doping in polysilicon layers,
- annealing contact regions,
- growing epitaxial layers from deposited amorphous films,
- annealing implanted device areas,
- relieving stress in silicon-on-sapphire or in local isolation regions.

## 2.4 DOPING

Ion implantation is steadily replacing solid and gaseous diffusion as the primary means of doping silicon because ion implantation offers much better areal doping uniformity (better than 1%) as well as profile tailoring. The uses of ion implantation for backside gettering, channel threshold adjustment, source/drain or emitter/collector doping, and resistor fabrication, are well known. Machine capability is steadily being improved, particularly for higher currents up to 10mA, and higher ion energies up to 400 KeV (8). These higher throughput machines give rise to concerns with heat dissipation in the wafer.

One trend in ion implantation is toward very low energy (less than 10 KeV) implants for shallow distributions which will be needed for threshold adjustment of micrometer-sized MOSFETs. The uniformity of low dose/low energy implants is still a significant problem. Another concern associated with such low energy implants is anomalous channeling which degrades the distribution. Yet another difficulty is grain boundary channeling, particularly of As or P through polysilicon gate electrodes (9). Another new implant application uses extremely high energies (2-3 MeV) to implant deep buried layers for alpha particle collection grids. Other novel applications of ion implantation include enhanced etching of oxide and silicon regions, contact via hole doping, silicide formation, and double-diffused (DMOS) FET's.

Focused ion beams (10) offer one means for combining additive, selective, and subtractive processing operations. Potentially, this technique could selectively dope the substrate, expose resist patterns, or sputter etch thin films. The goal is to reduce processing steps and eliminate masking operations. Narrow line widths and precise registration will be required, however. To date, in the research laboratory, Ga ion beams have been used for doping and machining selected regions. Focused boron and arsenic beams have also been demonstrated. The most likely initial application of focused ion beams is in special applications requiring customized fabrication.

## 2.5 METALLIZATION

By far, the most popular material for metallic low resistance interconnection lines in silicon integrated circuits is aluminum. Aluminum is abundant, inexpensive, easy to evaporate and to pattern, self-passivating, and adheres well to both silicon

and silicon dioxide. The most common metal deposition techniques are:

- evaporation from a RF heated source,
- evaporation by electron-beam heating, and
- sputtering.

Of these, the RF heating approach offers the least radiation or surface damage, particularly for FET fabrication. DC magnetron sputtering also has low associated radiation. The deposition of other metals for rectifying contacts (11) (e.g., Pt, W, Ta, Nb) often requires electron-beam heating with higher risks of radiation damage.

One of the most important trends in metallization is the development of two, three, or even four layers of metal wiring paths. Most of the difficulty centers around the insulating layers between metal levels (sputtered quartz, nitride, oxide-nitride or polyimide) and the contact holes through these layers. The insulating layers must be deposited at low temperatures so as not to degrade the first metal (aluminum) interconnections. As more layers are added, the topography of the structure becomes less and less planar, and correspondingly the linewidth control degrades. For example, in a triple metal system, the third level metal lines may have to be double the width of the first level lines. A goal is to develop planarizing techniques and via refill steps to improve planarity and line control. FETs with two metal levels and bipolars with three are now commonplace.

Aluminum, gold, and silver do not form intermetallic compounds with silicon, but many other metals such as Pt, Pd, Ti, Ta, Mo, and W do. These intermetallic compounds, called silicides, provide intimate metal-semiconductor contacts which have a

number of useful properties including: high barrier heights, high eutectic formation temperatures, and low resistance. Applications of silicides include both rectifying and ohmic contacts, and, potentially, interconnection lines on top of polysilicon or diffused regions. A new area is the use of ion or laser beams to form silicides.

Low formation temperature silicides like PtSi and Pd<sub>2</sub>Si have long been used as rectifying contacts and more recently studied as a means for reducing contact via resistance between Al lines and Si regions. Such techniques are applied late in the process after all high temperature steps have been completed. The development of silicide layers that can withstand high processing temperatures is now one of the most active research and development areas for IC metallurgy.

It has been proposed that a high formation temperature silicide such as tungsten, tantalum or molybdenum silicide could be used to reduce the sheet resistance of polysilicon or diffused silicon regions in FETs (12). For process groundrules of over three micrometers, diffusion depths and polysilicon thicknesses are large enough that sheet resistances of 15 to 30 ohms/square can be obtained. As dimensions are reduced, however, sheet resistances rise degrading performance. A major objective is to cover the thin polysilicon layer with a low resistance silicide layer yielding a composite "polycide" layer with the gate electrode properties of polysilicon but with a sheet resistance of one to five ohms/square (13, 14). The layered gate electrode/interconnection line material must withstand high temperature oxidation and annealing steps and be easy to pattern and etch selectively (13). The etching step is particularly troublesome as polysilicon and metallic silicides have quite different chemical behavior. To date a



simple preferred polycide technique has not been disclosed and a compromise between sheet resistance, etching behavior, and durability must be achieved. Polycides are also important for advanced bipolar structures that use thin polysilicon layers for wiring and as the source of shallow emitter doping (15).

## 2.6 PASSIVATION

A passivation layer is required over the silicon chip to inhibit damage from mobile ions (Na, Cu, etc.) and water vapor. Historically, sputtered quartz, phosphorus-doped glass, and oxide-nitride coatings have been used. More recently a move to plasma enhance CVD nitride and to polyimide has occurred (7). The primary attractions are lower cost and more conformal coatings.

Polyimide is particularly attractive in cases where multilevel metal wiring is employed. An interesting research development is a photosensitive polyimide which could eliminate resist masking and etching of the passivation layers. Organic coatings for absorption of package-generated alpha particles are now popular for dynamic memory chips.

## 3.0 SELECTIVE OPERATIONS

Selective operations involve the exposure and development of lithographic patterns into a photosensitive layer (e.g., a resist). Over the past 20 years, the lithographic improvement traced in Figure 2 has progressed from proximity to contact to projection printing, all with full wafer exposure. As indicated in Figure 4, other developments in lithographic technique are expected in the future. Presently, the manufacturing capability of full-field projection printing is about  $2.5 \pm 1.0$  micrometers. This lithographic groundrule

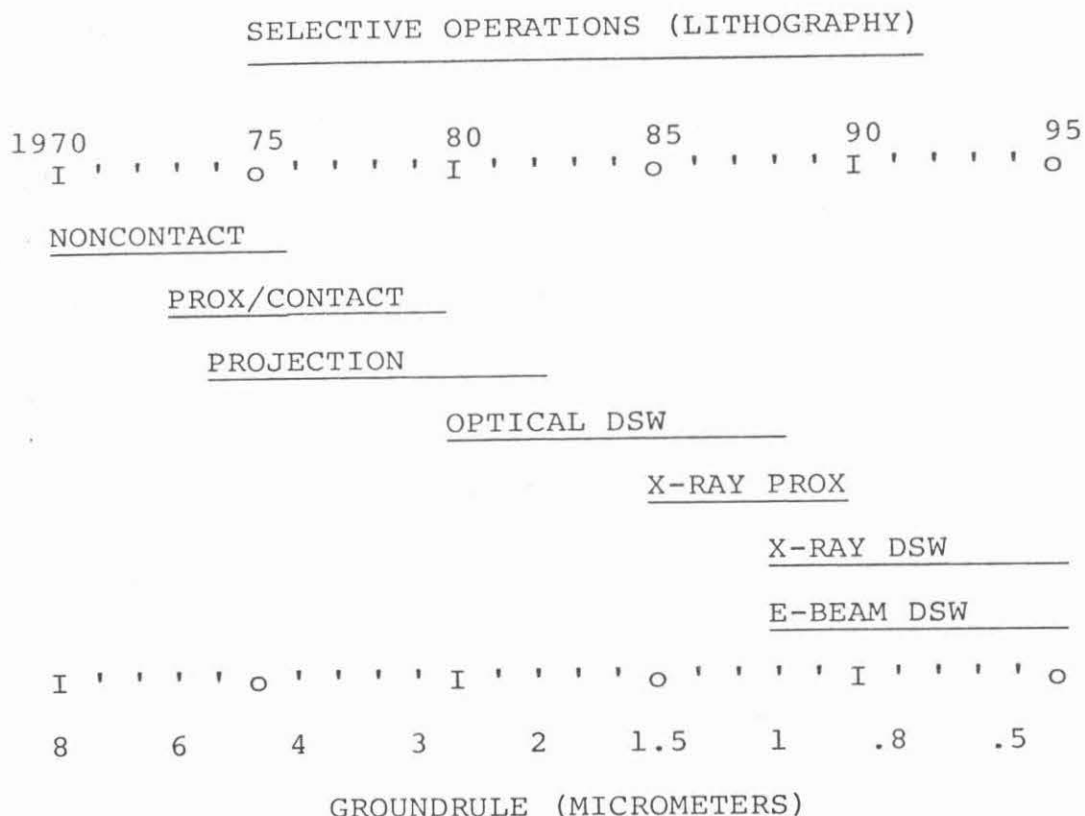


FIGURE 4: Development of lithographic exposure techniques.

refers to the resist patterns, not the final fabricated feature sizes on the silicon wafer which are approximate 1.5 times larger (16) (e.g., about 3.5 to 4.0 micrometers). A natural extension to shorter wavelengths (250 nm or "deep" UV) is occurring with the necessary transition from glass to quartz mask plates. When compared to standard UV (320 nm), deep UV enables the proximity gap between mask and wafer to be widened to reduce mask damage while maintaining system resolution. Alternately, operation with the same gap will give resolution increased by the square root of the wavelength ratio.

In order to progress to below 2 micrometers it appears that limited field, step-and-repeat (i.e., direct-step-on-the-wafer or DSW), projection optics with automation alignment will be required. Many new IC facilities now under construction are strongly dependent on optical step-and-repeat lithography. Linewidths of 1 to 1.25 micrometers and registration of  $\pm 0.4$  to  $\pm 0.6$  micrometers should be possible near the end of the decade.

Beyond one micrometer, a strong competition is developing between direct write electron-beam and projection X-ray. Although the outcome will not be decided for at least 5 years, both techniques still have serious deficiencies. Electron-beam lithography is costly, complicated, and in need of more robust and more sensitive resists. Electron-beam machines have, however, become widely accepted as mask makers for optical projection, and at IBM are also used in production for the customizing of final level metal patterns in bipolar logic arrays. X-ray lithography is less well developed and needs more energetic sources, stable masks, more sensitive resists, and an automatic alignment scheme. An interesting proposal in the lithography field is to develop a relatively smaller electron storage ring for the generation of intense X-rays (17). Such a ring would service several (e.g., up to 10) X-ray lithographic stations.

In the resist area, an important activity concerns conformal multilayer masking techniques (18, 19). This is a means for circumventing low resist sensitivity and depth of field constraints, and for improving resolution (14). As shown in Figure 5, a very high resolution pattern can be formed in a thin layer of resist which would be too thin to be used for etching. But this high resolution pattern can be transmitted down into a thicker layer of working resist,

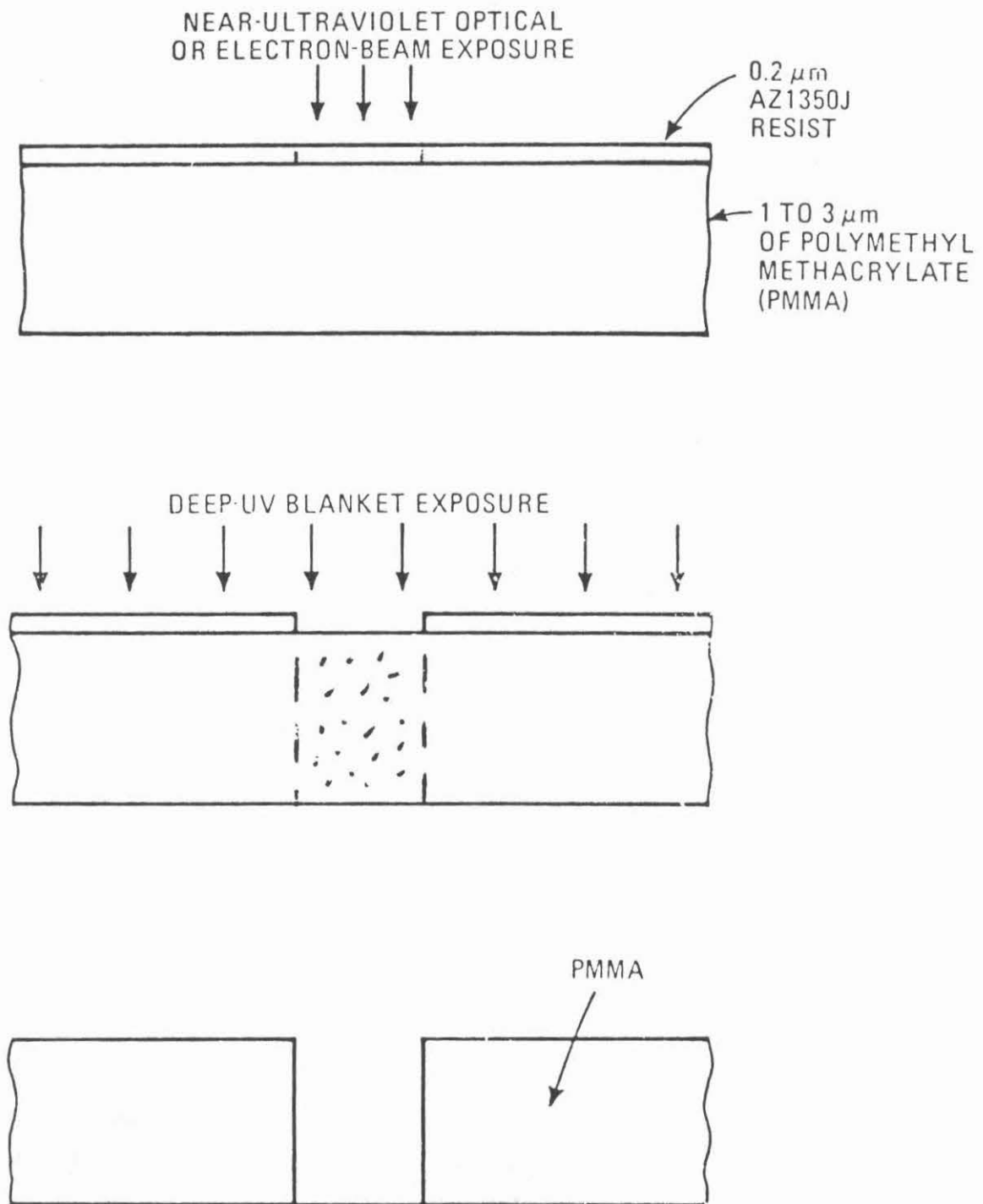


FIGURE 5: Portable conformal masking technique (after Ref.18).

without much loss in resolution, by a blanket exposure. IBM calls this portable conformal masking because the multilayer resist structure is transported from one exposure station to another and because the thicker resist conforms to the topography below it.

Over the past ten years, photolithographic linewidths have halved every 6 years (see Figure 2). The combined progress in lithographic machinery, resist materials, and etching techniques can double the chip density about every 4 years at best (see Figure 1). It is expected that improvements in optical projection lithography can sustain this rate of improvement for at least the next five years.

The transition to E-beam or X-ray lithography in production should take place in the latter half of the decade as optical wavelengths constrain photolithography to about one micrometer dimensions (see Figure 4). This transition will be slowed by the severe technical and economic difficulties inherent in introducing any new lithographic technology, by the problems associated with even larger chip and wafer sizes, and by the staying power associated with an immense investment in the highly utilitarian optical technology.

#### 4.0 SUBTRACTIVE OPERATIONS

The transfer into the substrate of the exposed and developed mask pattern in the photoresist layer is accomplished by a subtractive etching operation. Historically, such operations were carried out using wet chemical etchants such as hydrofluoric, sulphuric, or phosphoric acids. The attraction of wet etchants is that they are highly selective, generally attacking only one layer species. Unfortunately, they are

frequently isotropic or non-directional in nature and hence tend to etch under the masking layer. Wet etches are temperature and concentration dependent and overetching is often needed to insure complete material removal.

Important improvements have been made in recent years with etching in RF generated plasmas (20, 21, 22) which is sometimes referred to as dry etching. Figure 6 illustrates the difference between sputter, plasma, and reactive ion etching. In sputter etching, a non-reactive gas such as Ar is used and the sample is simply bombarded with directional, energetic ions. The etching, however, is indiscriminant, i.e., non-selective. Thus, there is no physical mechanism that stops the etching process when a second layer is revealed.

With plasma etching (20, 22) a reactive gas species like  $\text{CF}_4$  or  $\text{CCl}_4$  is used with the wafers at plasma or ground potential and a pressure of 0.5 to 2 Torr. The high gas pressure leads to a random incidence of etching species. These conditions and the wafer positioning lead to an isotropic (i.e., non-directional) but highly selective etching which is widely used in the industry today, for example for ashing (stripping) exposed resist layers and for etching thin  $\text{Si}_3\text{N}_4$  layers. As contrasted to the barrel assembly, the use of a parallel plate reactor improves the directionality of plasma etching, but with a loss in selectivity. Selectivity ratios of 10 or 20 to 1 are possible although 5 to 1 is more typical. The transition from wet etching to dry (plasma) etching is required for linewidth control for features in the 2 to 3 micrometer regime (14).

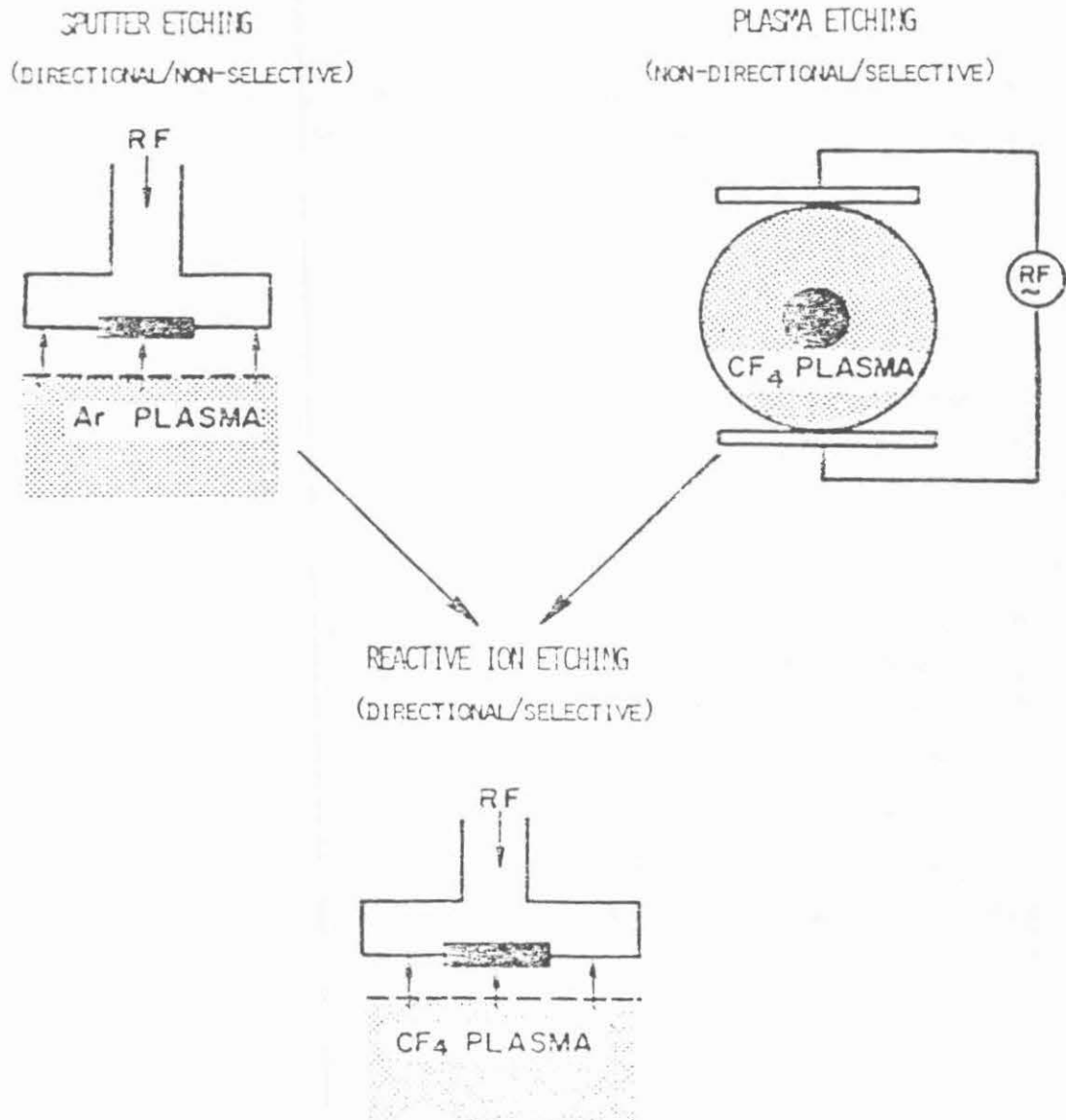


FIGURE 6: Comparison of sputter, plasma and reactive ion etching (after Ref. 22).

With reactive ion etching (RIE), a reactive gas like  $\text{CF}_4$ ,  $\text{CClF}_3$ ,  $\text{C}_2\text{F}_6$ ,  $\text{CCl}_4$ , etc., is used at lower pressures (.020 to .040 Torr) and with the wafers placed on the cathode (e.g., out of the plasma discharge region) (23). A directional and selective etching condition results. The etching behavior is sensitive to various parameters such as RF power, gas pressure, and the



choice of etching gas and cathode material. A strong attraction of RIE is that the etching gas composition can be manipulated to obtain different high etching rates (24). Also, by increasing the gas pressure or altering the sample position, combinations of reactive ion and plasma etching can be utilized (22).

The drawbacks to reactive ion etching are primarily technical (batch sizes, etch rate uniformity, understanding of the etching chemistry) and hence RIE will become more widely accepted as techniques improve. The combination of selectivity and directionality afforded by reactive ion etching is essential to the development of one micrometer processes. A particular need is for robust photoresists that are highly resistant to plasma or reactive ion etching. Another need is for the ability to provide a controlled slope on the edge of an etched line to relieve line coverage problems.

## 5.0 PROCESS MODELING

Computer aided design (CAD) tools are of increasing benefit to chip and circuit design, for example, for wire routing, cell placement, timing analysis, and design rule checking. Device modeling is also valuable in predicting electrical parameters, cutoff frequencies, threshold voltages, and so on. Only recently has process modeling begun to play an important role in integrated circuit fabrication (25).

Integrated circuit modeling activities can be roughly subdivided into process, device, circuit, and chip architecture areas. Device modeling using Poissons' equation and the continuity equations has been active for over twenty years, however process modeling dates back less than a decade. The primary purpose of process modeling is to provide a description of the device

structure that can be utilized in a device analysis model. This structural description may include impurity distributions, insulator thicknesses, and device dimensions such as channel lengths and widths. This activity can be referred to as process profile modeling. The device analysis model utilizes this profile information in predicting the device parameters such as current-voltage or voltage-voltage relationships. These electrical relationships can then be incorporated into a circuit simulation model to predict switching rates or signal propagation times. The result of such an analysis will be a description of the nominal circuit performance. Before attempting to fabricate a circuit however, one should also determine the statistical range of circuit performance.

Variations in the fabrication process are unavoidable. Some of these are natural in origin such as uncertainties in substrate resistivity, others are equipment related arising from variations in furnace temperatures and implantation doses, and yet others are due to operator differences in etching times or other procedural variables. The cumulative result of many small statistical variations in fabrication can induce a significant resultant error in the final electrical parameters. It is here that process control modeling can be of value. Deviations from the base line process conditions can be deliberately introduced into the model and a sensitivity analysis performed (26). This can help to determine if one of the process steps is close to a critical point. In one instance in the author's experience, process models showed that the chosen implanter energy was on a steep sensitivity slope. Reducing the energy and increasing the dose gave a safer fabrication condition which had less effect on the range of the resultant electrical device parameters.

Relative to CAD for circuit development, process modeling is still in its infancy. The process work began as one-dimensional profile analysis, by either analytical or numerical models, and progressed to two and three-dimensional forms. The initial work concerned primarily the additive materials operations. More recently, the selective operations of resist exposure and development have been addressed with the intent of defining linewidth parameters (27). More work still remains to be done in modeling of oxidation and diffusion which are now well developed and suitable for computer modeling. Plasma and reactive ion etching, however, are less well understood which complicates the description of their behavior. In the area of profile modeling, two successful efforts concern descriptions of channel stoppers for FET isolation (28), and double-diffused regions for bipolar emitters (29).

Until very recently process modeling has been mostly an "after-the-fact" activity. Typically, process modeling was brought into play only after difficulties were detected in a new process, or in a modified older process. The VLSI era promises to be different. First, strictly from an economic point of view, the cost of experimental pilot line facilities is so high that brute force trial-and-error process development is too expensive. Thus tightly coupled process and device modeling activities will be required to optimize the process development cycle. Second, more highly integrated circuits promise to be more sensitive to statistical variations whether natural in origin or introduced during fabrication. Statistical fluctuations in doping, high densities of small defects, layer thickness control, and a host of problems may plague VLSI fabrication in the micrometer and submicrometer lithographic regime. Consequently, two and even three dimensional process control models will be required to help identify, understand,

and avoid costly fabrication problems. Third, novel process steps can now be investigated with a computer model. This gives the process engineer a new tool for innovation.

#### 6.0 AUTOMATED PROCESSING

Process automation is slowly and steadily being incorporated into integrated circuit fabrication. Automation can be applied in several ways (30). One area is robotics, or automated wafer handling in which the wafers are moved on air tracks (rather than by operators) and are mechanically inserted into open mouthed diffusion and evaporation stations. Two such facilities exist within IBM. A second area is on-line inventory control, also used at IBM, in which the position of various wafer lots in the line is determined and monitored by computer. This is especially important for high throughput manufacturing. A third area of automation is local process control in which, for example, microprocessors are used to ramp furnaces or insert wafers at a particular station.

An important adjunct to process automation is on-line monitoring of gas purity, furnace temperature, etc. This activity is highly transducer dependent and much progress can still be made here. An interesting hypothesis is that real time process information could be used to modify a step further along in the process. For example, if the monitor showed that the gate insulator thickness was above nominal, the subsequent channel implantation dose could be reduced accordingly. Such on-line process tailoring might be required for the most sophisticated VLSI fabrication. The objective of process automation is improved process quality or process throughput, or both. Probably the simplest and most direct way to increase productivity, however, is to increase the wafer size. Over the past decade wafer diameters have

increased from one to five inches. By 1990, we can expect wafer diameters of seven to nine inches. Silicon strips or ribbons may also be developed. Each move to a larger wafer diameter is traumatic due not only to equipment changes, but due to size related problems as well. As wafers increase, thermally induced wafer stresses (warping), stress-induced dislocations, global and local distortions, and other problems arise. Improved lithographic dimension is an alternative productivity enhancer, and it is lithographic machinery development that experiences the most demanding requirements with each quantum jump in wafer diameter. The difficulties associated with increasing wafer size tend to constrain progress in lithographic dimension. Compared to full field exposure, direct step-on-the-wafer, whether optical, E-beam, or X-ray, is relatively less sensitive to increased wafer size due to the limited field exposure area.

## 7.0 DEVICE STRUCTURES

The intended result of new processing techniques is an integrated transistor device structure that is smaller, cheaper, faster, lower in power, and more reliable than its predecessor. The basic structural elements of an integrated circuit are electrical isolation (between devices), the active device itself, contact vias (layer intraconnection points), wiring (interconnection lines), and passivation.

### 7.1 MOSFET TECHNOLOGY

Figure 7 shows a high density N-channel MOSFET or NMOS structure typical of the industry today (31). The structure is characterized by semi-recessed oxide isolation, a polysilicon-gate FET, an etched and rediffused contact area, polysilicon

and diffusion and aluminum wiring paths, and phosphorous-doped glass passivation.

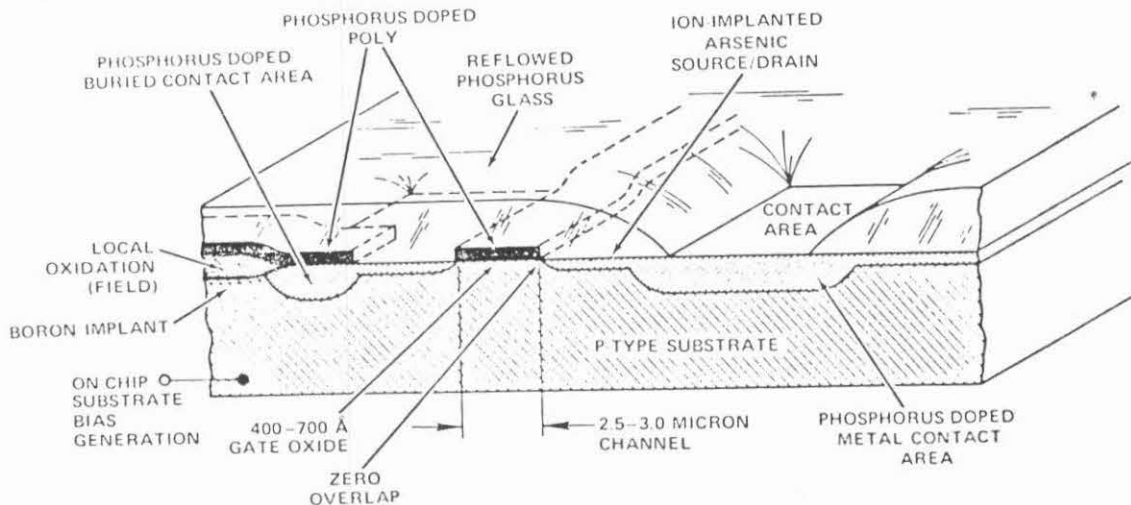


FIGURE 7: Present MOSFET structure (after Ref. 31).

### 7.1.1 ISOLATION

In MOSFET technology, dielectric oxide isolation between devices has progressed from planar (global thick oxide with etched holes) to locally grown (semi or fully recessed). Semi-recessed oxide is the mainstay of polysilicon-gate MOSFET technology due to its simplicity of fabrication, higher density, and self-aligned parasitic channel stopper. The locally grown or recessed oxide techniques require an oxidation-resistant silicon nitride layer. Oxide growth gives rise to a lateral oxidation wedge under this layer shaped like a bird's beak which reduces the active device area and complicates the structure (32). Although lacking high surface planarity, a simple approach is to use a common channel and field boron doping thereby eliminating nitride from the isolation step (33). This technique becomes more attractive at smaller dimensions with thinner isolation layers. Ideally, one would eliminate

silicon nitride from the process and yet provide a deep, narrow, oxide region flush with the substrate surface and possessing a doped channel stopper region.

Complementary (CMOS) FET technology has a special problem in that the isolation must help prevent PNP (silicon-controlled rectifier) latchup. A common approach is to widely separate devices and use the inherent diffusion isolation of the structure. Sapphire substrates provide complete dielectric isolation, which greatly improves the density, but the cost and processing difficulties of sapphire (e.g., outdiffusion, epi control, etc.) have led to a declining interest in these substrates. The idealized deep dielectric isolation discussed above could greatly benefit the bulk CMOS technology.

#### 7.1.2 DEVICE STRUCTURES

Two of the most interesting developments in MOSFET devices are the double-diffused or DMOS device (34), and the lightly-doped drain MOSFET (35). Figure 8A illustrates the DMOS device. The idea is to mask the drain and to laterally diffuse in a narrow p-type channel region on the source side of the FET. By this method, for example, a 0.5 micrometer channel length can be fabricated with 2.0 micrometer lithography. The advantages are higher gain, faster switching, and better channel length control. The penalties are added cost (one additional masking operation) and unilateral device operation. Static and dynamic RAMs and uncommitted logic arrays have been made using DMOS devices, but it is yet to be established that the performance improvement warrants the additional processing complexity.

In the lightly-doped drain structure shown in Figure 8B, the electric field at the drain is reduced by grading the diffusion profile there (35). Consequently, the drain voltage may be increased, thereby increasing the switching speed and/or current carrying capability. Extra masking steps are not required as the device fabrication is based on a controlled oxide overhang.

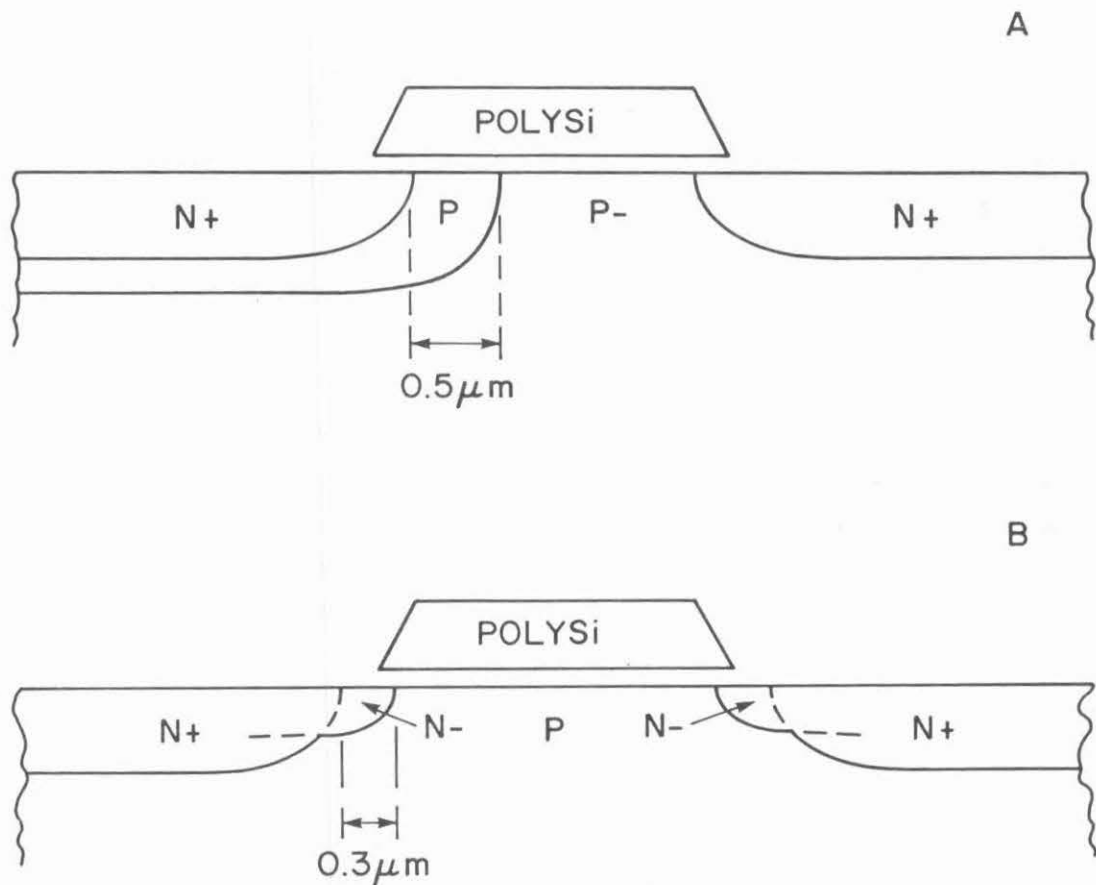


FIGURE 8: DMOS device (A) and lightly-doped drain FET (B) (after Refs. 34 and 35 respectively).



### 7.1.3 CONTACT VIAS

Contact resistance promises to be one of the first major problems to be encountered with higher integration because the resistance increases even more than linearly as contact area decreases (36). This antiscaling behavior is combatted, to first order, by deeply rediffusing the contact hole (37) (see Figure 7). Widely used in production, this technique is referred to as a borderless contact, but actually the diffused area is expanded under the isolation oxide by a rediffusion step. Another approach to optimizing metal to diffusion or to polysilicon contact areas is the self-registering contact in which an oxide layer is locally grown up around a nitride protected contact area (38).

Direct polysilicon to diffusion contacts buried under thick oxide (i.e., "buried contacts") become more difficult to fabricate as processes are scaled down because thermal drive times are reduced and a thinner polysilicon layer contains less N-type dopant. New approaches will be needed here.

### 7.1.4 WIRING

The most important advances in MOSFET wiring are double level metal and silicide-on-polysilicon (polycide) interconnections. N<sup>+</sup> diffused regions can also be used for wiring, however, with scaling the higher sheet resistance of shallower diffused lines discourages it. Although it requires at least two extra masks for contact via and wiring patterns, double level metal is finding acceptance in FET production. The advantages include density improvements in 4 Kbit quasi-static RAMs, lower resistance wiring in CMOS microprocessors, and increased yield with redundancy wiring for 64 Kbit dynamic RAMs.

The refractive silicide on polysilicon technique offers 5 to 10 times lower line resistance without additional masking operations (13, 14). The processing difficulties with incorporating a reproducible W, Ta or Mo silicide layer into the existing FET process are formidable, however, this technique should be available in the near future. The reduced interconnection line resistance will lead to higher speed operation, especially in static RAMs and in microprocessors.

The use of an intermetallic silicide layer to reduce the sheet resistance of polysilicon or diffused region wiring represents one of the most important processing trends today. Although the polycide wiring approach has been demonstrated with both bipolar and FET test vehicles, thus far it has not been incorporated into mass manufacturing due to fabrication difficulties. When used on top of polysilicon, the silicide layer must be essentially transparent to the process, that is it must be patterned and oxidized along with the underlying polysilicon layer. This is not a trivial requirement. Generally silicide layers are more resistance to plasma and wet etches, and can become brittle when oxidized. Also, the silicide may fail to adhere to the polysilicon layer.

The most popular approach is to use a silicide layer over a polysilicon layer, the idea being to simultaneously retain the favorable properties of the polysilicon either as a gate electrode material in FETs or as a diffusion source and contact for bipolar emitter or base regions, and incorporate with it the low sheet resistance of the intermetallic silicide. A silicide layer alone would be easier to pattern but has poor oxidation properties and cannot serve as a controlled diffusion source. The most popular silicides being investigated are the high temperature ones like  $\text{WSi}_2$ ,  $\text{TaSi}_2$ ,  $\text{MoSi}_2$ ,

$\text{NbSi}_2$ , and  $\text{TiSi}_2$ . The primary deposition techniques are sputtering, or co-evaporation by electron beam. Following the deposition, the silicide molecule must be established, or formed, by heating the composite at an elevated temperature (e.g.,  $900^\circ\text{C}$  for 30 minutes). The patterning may be done either before or after the forming step.

An interesting new technique is the use of ion implantation (39) or laser annealing (40) to form the silicide. In the former case the deposited silicide layer is bombarded by an arsenic beam, the energy dissipation of which causes the silicide layer to form, thus eliminating the high temperature heating step.

It is interesting to speculate that the ion beam annealing technique for silicides might also be used to anneal, for example, implanted source and drain regions, possibly with an argon beam. Of course, lasers or electron beams can be incorporated into ion implanters. Overall, there is a constant desire to combine processing steps into situ, although thus far little of this has occurred. There still is a great deal of wafer handling involved in an IC process which may require as many as 150 sequential operations.

#### 7.1.5 MOSFET

Figure 9 shows a hypothetical IC MOSFET of 1990. It is a polysilicon-gate bulk CMOS structure with one layer of polysilicon with silicide over it. Very shallow and lightly doped source and drain regions are used with laterally diffused regions for threshold control. Two layers of metal wiring are employed. The topography of the structure is highly planarized due to the fully recessed, deep dielec-

tric, field isolation and the planarizing passivation layers. High conductivity silicide layers over the diffused and polysilicon regions greatly improve the electrical conductivity. Contact vias to connect metal lines to metal, diffused, and polysilicon lines are refilled with conductive material. The vias are self-registering to the lines they contact. The channel length of the device is  $1.0 \pm 0.25$  micrometers and the threshold voltage is 0.5 volts. The gate insulator thickness is 250 Angstroms.

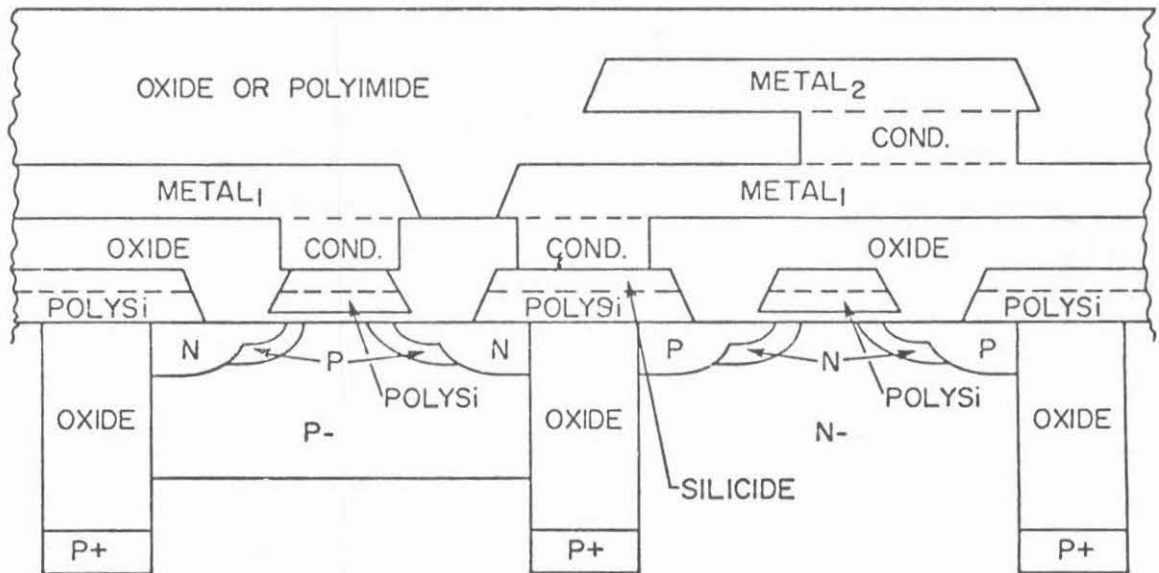


FIGURE 9: Future FET structure.

## 7.2 BIPOLAR TECHNOLOGY

Recently, the major emphasis in MOSFET development has been on dimensional reduction rather than on structural innovation. In contrast, bipolar technology is going through a minor renaissance in structural improvement. Just as the emergence of polysilicon gate electrodes spurred advances in FETs, the invention of integrated-injection logic has inspired innovation in bipolar device structures. Additionally, fabrication advances in FETs have influenced bipolar design which now include, for example, self-aligned regions, polysilicon doping, and polycide interconnections.

### 7.2.1 ISOLATION

Bipolar isolation progressed directly from diffusion isolation to fully recessed oxide. Figure 10 shows a cross-section of an IBM masterslice bipolar logic structure (41). Planarity is a key requirement as the substrate must support three levels of metal wiring for which the linewidth control is affected by surface topography. The idealized deep, narrow dielectric isolation described for MOSFETs would, of course, also benefit bipolars. Historically, novel isolation schemes have been more readily accepted into bipolar processes.

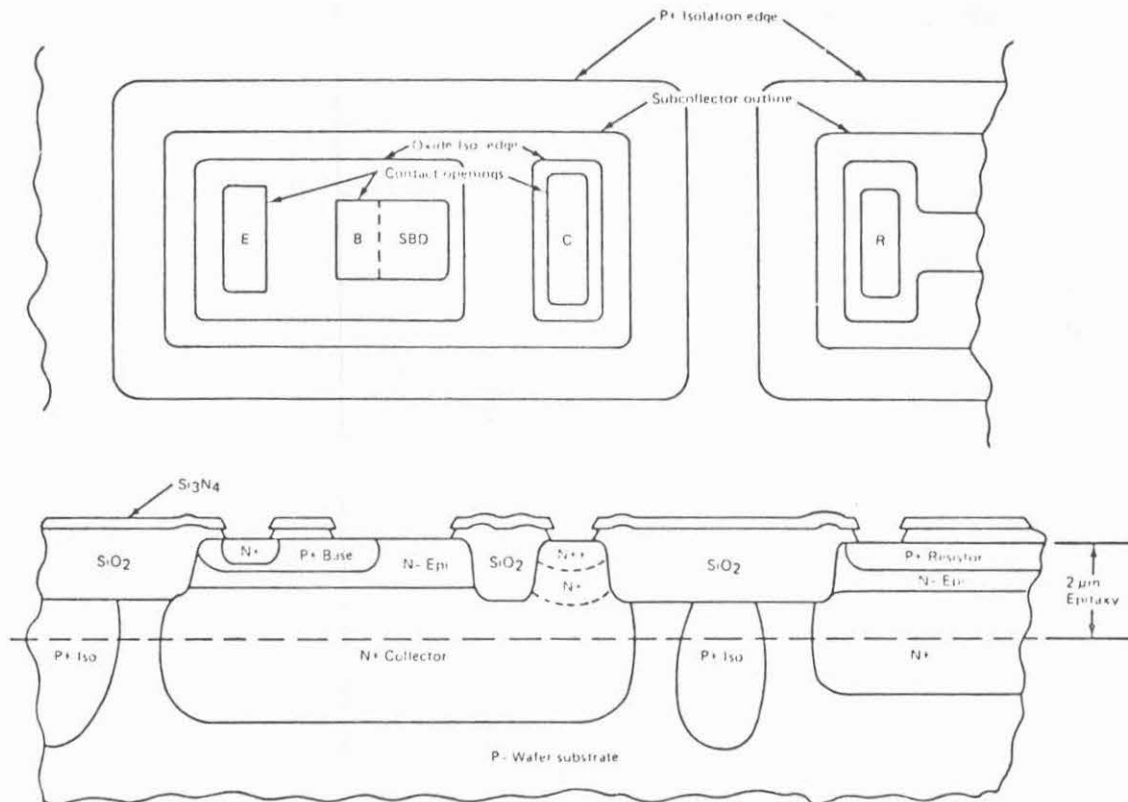


FIGURE 10: Present bipolar structure (after Ref. 41).

### 7.2.2 DEVICE STRUCTURE

A high degree of novelty is being incorporated into new bipolar structures. Among the most important of these are collector regions doped from and contacted by polysilicon, self-aligned collector-base contact edges, collector region that abut the isolation oxide areas, and metal-interconnected base regions (42). A major thrust is toward 0.1 micrometer base widths facilitated by limited outdiffusion. Another is toward reducing parasitic capacitances by reducing collector area (butted regions) and by moving the base contact closer to the active base region (self-alignment).

### 7.2.3 CONTACT VIAS

To date, conventional etched contact vias have been used to emitter, subcollector, and extrinsic base regions. The use of polysilicon, or silicide-on-polysilicon, allows aluminum lines to contact polysilicon regions over the isolation regions. This relieves aluminum spiking problems and reduces the overall device area.

### 7.2.4 WIRING

Bipolar logic structures today use three levels of metal wiring (41) and this might increase to four or five levels in the future. The use of polycide layers, however, may alter this trend. The attraction of multilevel metal is high conductivity and low temperature processing, the drawbacks being extra masking steps and increasingly larger groundrules for the upper levels. Conductive metal refill techniques for contact vias between metal levels are needed, as are self-alignment techniques to register holes to lines.

### 7.2.5 FUTURE STRUCTURE

Figure 11 shows a hypothetical bipolar transistor of 1990. It is a T<sup>2</sup>L structure with silicide-on-polysilicon for the emitter, base and collector doping. Device regions butt up against the ideally deep dielectric isolation. Three layers of metal wiring are employed. Contact vias between metal layers are refilled with metal and self-registering contact techniques are used. The base width of the transistor is 0.1 micrometers. Like the future FET, the bipolar structure is highly planarized to relieve line coverage problems and reduce linewidth of upper metal layers.

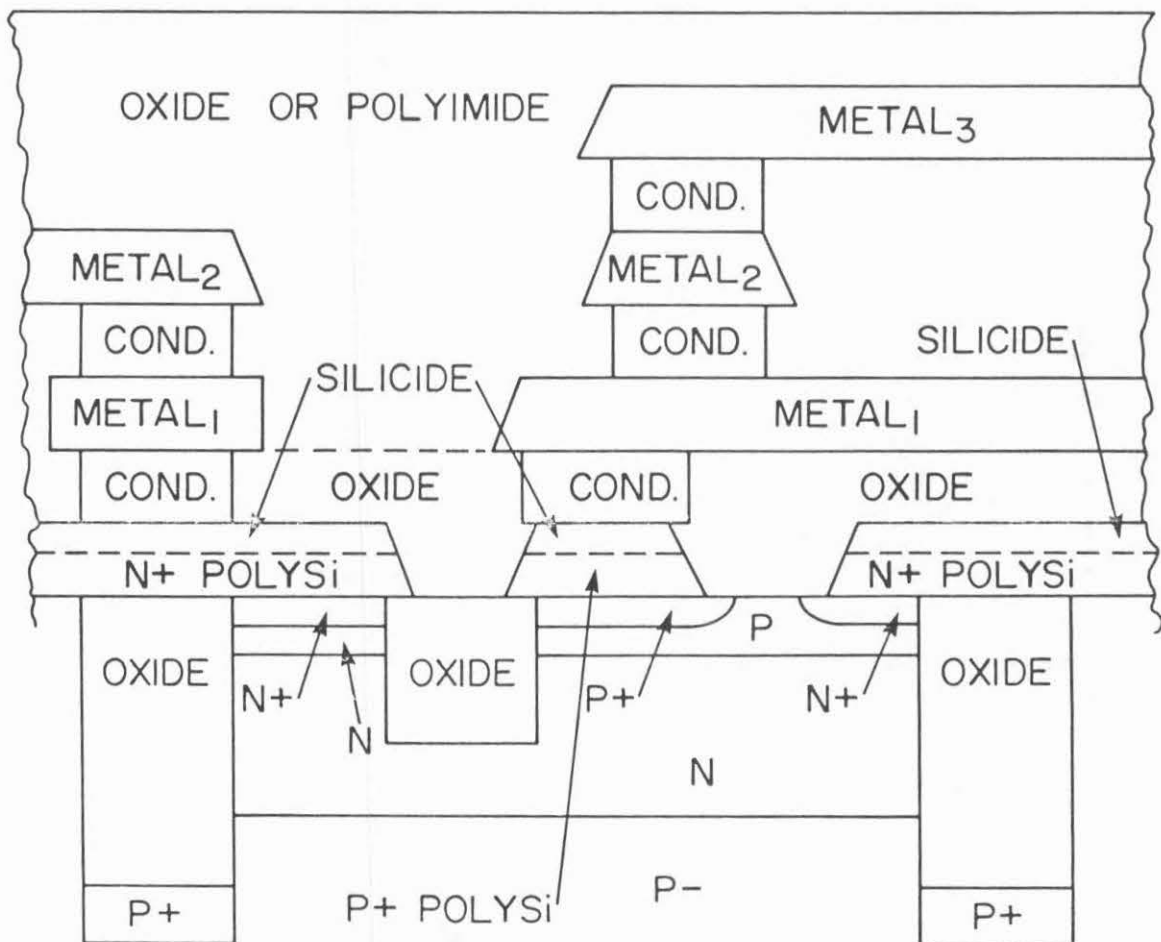


FIGURE 11: Future bipolar structure.



## 8.0 STRUCTURAL PROBLEMS

### 8.1 ISOLATION AREA

Over the past decade, reduction in isolation area relative to device area has led to significant density improvements. This is exemplified by the transition from diffusion isolation to fully-recessed oxide isolation in bipolars, and from thick oxide with non-registered channel stoppers to semi-recessed oxide with self-aligned channel stoppers in N-channel MOSFETs. Nevertheless, today about 50% of an IC chip area is still devoted to isolation. A major improvement in isolation is needed for density enhancement. The idealized deep and narrow dielectric isolation, which we hypothesized earlier, might reduce isolation area to 25%. The isolation dielectric could be oxide, nitride, polysilicon, or combinations of these materials, and the refill should be planar with respect to the substrate surface.

### 8.2. CONTACT RESISTANCE

Contact resistance is one of the parameters that defies scaling and promises to present a major difficulty for VLSI (1). As contact areas are reduced, contact resistances increase linearly or superlinearly with contact diameter. One micrometer contact diameters with 10 to 100 ohms contact resistance can be expected (36). Techniques to insure uniformity such as rediffused (37) or laser annealed (40) contact holes help reduce contact resistance which is determined by the area, thickness, and resistivity of the contacted region. In this regard, an aluminum to silicide-on-polysilicon contact becomes particularly attractive.

### 8.3 LINE COVERAGE

The ability to cross one conductive line with another is impaired if the edges of the lines are not sloped or if the intervening insulating layer is not conformal. The problem is manifested in presentday products in aluminum lines crossing dry etched polysilicon lines in which a reflowed phosphorus-doped glass insulator is not used. Line thinning or breaks can occur at the crossing points. This will present an additional burden to the plasma etching techniques which will have to provide controlled slopes during delineation.

### 8.4 RADIATION

Radiation introduced during fabrication can detrimentally affect integrated circuits. Deposition, etching, and lithographic equipment is particularly suspect, especially for thin MOSFET gate insulators (43). Although it appears that any radiation damage that might be introduced by chemical vapor deposition steps or plasma etching can be annealed out at subsequent processing operations at over 800°C, metal deposition presents a different situation. Aluminum, for example, is deposited in vacuum by RF heating, by sputtering, or by electron-beam heating. The latter technique involves considerable radiation damage which cannot be annealed out at high temperatures as aluminum melts near 500°C. Electron-beam (or X-ray) lithography also introduces radiation damage, which, if used for the final metal definition may be hard to remove (44). Plasma etching can also introduce radiation damage (45).

A deleterious effect of process radiation is to produce damage states (traps) in the thin gate insulator which may charge up during the operating life on the device. Thus a

device parameter (e.g., the threshold voltage) may slowly move out of specification during the life of the device. One trend is to try to improve low temperature (below 500°C) annealing techniques by RF annealing or by thermal annealing in pure hydrogen. The area of process induced radiation damage is a new one that promises to be of much greater importance for VLSI fabrication.

## 9.0 SUMMARY

A variety of new fabrication techniques and structural elements have been reviewed thus far in this work. We have speculated as to how these trends in silicon processing might come together to produce the integrated circuits of the future. In particular we have tried to imagine what the silicon MOSFET and bipolar transistors will look like ten years hence and what techniques will be required to fabricate them.

### 9.1 IC FABRICATION OF THE FUTURE

Based on present projections, by 1990, one micrometer lithography with mask-to-mask alignment capability of  $\pm 0.25$  micrometers will be practiced in mass production. Most likely this will be achieved with optical direct-step-on-wafer (DSW) projection systems. MOSFET memory chips with one million components will be available yielding 128 Kbit static RAMs and 256 Kbit dynamic RAMs in production with developmental chips of twice that capacity just emerging. Furthermore, 64 bit FET microprocessors with 250,000 components will also be available with the equivalent computing power of 50,000 logic circuits. High performance (less than 10 nanoseconds) bipolar cache memory chips of 32 Kbits will be available. Bipolar masterslice logic will have 10,000 circuits (about 50,000

components) and denser bipolar circuitry (PLAs and semi-custom chips) will reach 100,000 components. Chip power dissipation will play a major role in determining that the leading MOSFET technology will likely be polysilicon-gate bulk CMOS while the bipolar technology will be low power T<sup>2</sup>L, or I<sup>2</sup>L. Key improvements in packaging technology will also occur.

Ten years from now, the IC fabrication process will use ion implantation for all doping steps and dry etching for all material removal steps. Polycide layers will be used for gate electrodes, wiring, contacts, and controlled doping of shallow regions. Combinations of plasma and reactive ion etching will be used to achieve the required degree of selectivity, directionality, and line shape. Multilayer resists will be available to withstand the plasma etching. Seven inch wafer diameters and chips as large as 100 mm<sup>2</sup> will be processed. Low pressure and plasma assisted CVD will be commonplace, as will high pressure oxidation. Scanned laser beams will be used for various annealing and forming steps. Process control and monitoring will be highly automated so that operators will be employed primarily for maintaining and repairing equipment or moving containers of wafers from one station to another. Monitoring information will be processed in real time so that later process steps may be customized to accommodate variations in earlier steps. The process line of tomorrow will look and operate somewhat like the computer center of today.

Technological prediction is an unreliable and highly imprecise art. In 1970, device and process researchers could not have predicted the HMOS polysilicon-gate FET or I<sup>2</sup>L bipolar structures of today, and fabrication techniques like plasma etching and laser annealing were then unknown. In a ten year timeframe,

lithographic dimensions decreased from over 6 to about 2.5 micrometers, wafer diameters increased from 1 to 5 inches, and device structures underwent revolutionary changes. In the next decade groundrules will decrease from 2.5 to 1 micrometer, wafer diameters will increase from 5 to 7 inches, and device structures will undoubtedly again undergo revolutionary changes. About the only certain future characteristic of integrated circuit technology is its unpredictability.

## 9.2 THE TWO CULTURES

The advances in very large scale integrated circuit design will be accompanied by substantial progress in microcircuit fabrication. This paper has concentrated almost exclusively on trends in silicon wafer fabrication techniques, only mildly considering lithographic progress, and ignoring completely requirements in packaging and in circuit and chip design. Obviously progress on all fronts will be necessary.

A challenging aspect of VLSI is the range of its impact upon the electronics industry. For example, digital technology ranges from solid state physical effects, through processing, devices, circuits, and chip architecture to system design. Two camps or cultures may be identified: chip fabricators that work in the semiconductor "foundry" handling silicon wafers and system designers that work in the CAD "foundry" handling terminal keyboards. The device and circuit designers work in the intermediate region between these two extremes. Clearly the coming of VLSI has required people with widely differing skills to work together.

The interface between the two cultures is often a difficult one. This is partly because the system tends to work from the top down with the design ideas driving the fabrication capability. Consider for example that a fabricator needs a

ten million dollar laboratory and fifty associates to do a one micron feasibility study, while the designer can simulate an entire chip right on his own computer terminal. Apparently the role of the lone innovator has shifted from the laboratory to the office. Certainly a very innovative and excited atmosphere exists today in the design world.

In the future fabricators and designers will have to work closely together and there is a tremendous range of technology to span. One small way of improving the fabrication-design interface and encouraging process innovation is to promote process modeling and automation. VLSI brings the two cultures closer together which in itself may be one of the most important future trends in integrated circuits.

#### 10.0 ACKNOWLEDGEMENTS

The author is partially indebted to S.C.Su. of the Hughes Research Center for making available information on low temperature processing. Discussion and helpful suggestions were also received from the author's colleagues at IBM including: J. M. Aitken, E. Bassous, J. M. Blum, L. M. Ephrath, W. D. Grobman, R. D. Isaac, B. J. Lin, L. M. Terman, and M. Y. Tsai.

#### 11.0 REFERENCES

1. V. L. Rideout, "Limits to Improvements of Silicon Integrated Circuits," Proceedings of Microcircuit Engineering 79, pp. 144-152, Aach. (September 25-27, 1979).
2. A. B. Glaser and G. E. Subak-Sharpe, Integrated Circuit Engineering, Addison-Wesley Pub., Reading, Mass. (1977).

3. Courtesy of S. C. Su, Hughes Research Center, Newport Beach, Calif.
4. R. J. Robinson, "CVD Process Trends," *Semiconductor Internat.*, pp. 27-37 (March, 1979).
5. P. E. Luscher, W. S. Knodle, and Y. Chai, "Automated Molecular Beams Grow Thin Semiconductor Films," *Electronics*, pp. 160-168 (August 28, 1980).
6. Work reported by Hughes Research Laboratories, Malibu, Calif.
7. P. S. Burggraaf, "Plasma Deposition Production Trends," *Semiconductor Internat.*, pp. 23-34 (March, 1980).
8. R. J. Robinson, "Ion Implanters Overcoming Current Barriers," *Semiconductor Internat.*, pp. 45-53 (June, 1979).
9. Y. Wada, S. Nishimatsu, and N. Hashimoto, "Arsenic Ion Channeling Through Single Crystal Silicon," *J. Electrochem. Soc.*, Vol. 127, pp. 206-210 (January, 1980).
10. R. L. Seliger and P. A. Sullivan, "Ion Beams Promise Practical Systems for Submicrometer Wafer Lithography," *Electronics*, pp. 142-146 (March 27, 1980).
11. V. L. Rideout, "A Review of the Theory, Technology, and Applications of Metal-Semiconductor Rectifiers," *Thin Solid Films*, Vol. 48, pp. 261-291 (1978).

12. V. L. Rideout, "Reducing the Sheet Resistance of Polysilicon Lines in Integrated Circuits," IBM Tech. Disc. Bul., Vol. 17, p. 1831 (1974).
13. B. L. Crowder and S. Zirinsky, "1 $\mu$ m MOSFET VLSI Technology: Part VII -- Metal Silicide Interconnection Technology -- A Future Perspective," IEEE Trans. Electron Dev., Vol. ED-26, pp. 369-371 (April, 1979).
14. J. Lyman, "Scaling the Barriers to VLSI's Fine Lines," Electronics, pp. 115-126 (June 19, 1980).
15. T. Takahashi, S. Wakamatsu, and K. Kimura, "A High Speed Multiplier Using Subnanosecond Bipolar VLSI Technologies," Eur. Solid-State Cir. Conf. Tech. Dig., pp. 110-112, Southampton (September, 1979).
16. V. L. Rideout, "Development of One-Device Random Access Memory Cells: A Tutorial," IEEE Trans. Electron Dev., Vol. ED-26, pp. 839-852 (June, 1979).
17. W. D. Grobmann, "Synchrotron Radiation X-ray Lithography," to be published.
18. B. J. Lin, "Portable Conformable Mask -- A Hybrid Near-Ultraviolet and Deep-Ultraviolet Patterning Technique," SPIE, Vol. 174, Develop. in Semicond. Microlitho. IV, pp. 114-121 (1979).
19. B. J. Lin and T. H. P. Chang, "Hybrid E-beam/Deep UV Exposure Using Portable Conformable Masking (PCM) Technique," J. Vac. Soc. Technol., Vol. 16, pp. 1669-1771 (November/December, 1979).



20. P. S. Burggraaf, "Plasma Etching Technology, "Semi-conductor Internat., pp. 49-58 (December, 1979).
21. C. J. Mogab and W. R. Harshbarger, "Plasma Processes Set to Etch Finer Lines with Less Undercutting," *Electronics*, pp. 117-121 (August 31, 1978).
22. L. M. Ephrath, "Dry Etching Review," invited paper to be presented at Silicon Symposium, Electrochem. Soc. Spring Meeting, Minneapolis (May, 1980).
23. L. M. Ephrath, "Reactive Ion Etching for VLSI," invited paper to be presented at IEEE Internat. Electron Dev. Meeting, Washington, D.C. (December 8-10, 1980).
24. L. M. Ephrath, "Selective Etching of Silicon Dioxide using Reactive Ion Etching with  $\text{CF}_4\text{-H}_2$ ," *J. Electrochem. Soc.*, Vol. 126, pp. 1419-1421 (August, 1979).
25. J. Meindl, et al, Process Models for Integrated Circuits, Stanford Univ., to be published.
26. D. A. Antoniadis and R. W. Dutton, "Models for Computer Simulation of Complete IC Fabrication Process," *IEEE Trans. Electron Dev.*, Vol. ED-26, pp. 490-500 (April, 1979).
27. A. R. Neureuther, D. F. Kyser, and C. H. Ting, "Electron-beam Resist Edge Profile Simulation," *IEEE Trans. Electron Dev.*, Vol. ED-26, pp. 686-692 (April, 1979).

28. V. L. Rideout, B. L. Crowder, and F. F. Morehead, "Implanted Boron Channel Stoppers for MOSFET Integrated Circuits," talk presented at IEEE Semicond. Interface Specialists Conf., New Orleans (December, 1979).
29. R. Reif, R. W. Dutton, and D. A. Antoniadis, "Computer Simulation in Silicon Epitaxy," J. Electrochem. Soc., Ext. Abstracts, Vol. 79-1, pp. 352-355 (May 6-11, 1979).
30. See special session on Process Monitoring and Automation at the Electrochem. Soc. Meeting in St. Louis (May, 1981).
31. M. Eklund, "IC Technology in the Eighties," Semicond. Internat., Vol. 3, pp. 29-38 (January, 1980).
32. E. Bassous, H. M. Yu, and V. Maniscalco, "Topology of Silicon Structures with Recessed Silicon Dioxide," J. Electrochem. Soc., Vol. 123, pp. 1729-1737 (November, 1976).
33. R. H. Dennard, and V. L. Rideout, "Method of Fabrication for Field Effect Transistors Having a Common Channel Stopper," U. S. Patent 4,090,289 (May 23, 1978).
34. Y. Tarui, et al, "Diffusion Self-aligned MOST -- A New Approach for High Speed Devices," in Proc. First Conf. Solid-State Devices (Suppl. to J. Japan Soc. Appl. Phys., Vol. 39, pp. 105-110, 1970).
35. S. Ogura, P. J. Tsang, W. W. Walker, D. L. Critchlow, and J. F. Shepard, "Lightly Doped Drain MOSFET Structure to Relieve Scaling Limitations," IEEE Workshop on Scaling and Lithography, N. Y. City (April 22, 1980).

36. H. Nozawa, S. Nishimura, Y. Horiike, K. Okumura, H. Jizuka, and S. Kohyama, "High Density CMOS Processing for a 16 Kbit RAM," IEEE Internat. Electron Dev. Meet. Tech. Digest, pp. 366-369, Washington, D.C. (December, 1979).
37. W. G. Watrous, "MOSFET Transistor and Method of Fabrication," U. S. Patent 3,986,903 (October 19, 1976).
38. V. L. Rideout, J. J. Walker, and A. Cramer, "A One-device Cell Using a Single Layer of Polysilicon and a Self-Registering Metal-to-Polysilicon Contact," IBM J. Res. Develop., Vol. 24, pp. 339-347 (May, 1980).
39. M. Y. Tsai, C. S. Peterson, F. M. d'Heurle, and V. Maniscalco, "Refractory Metal Silicide Formation Induced by As<sup>+</sup> Implantation," Appl. Phys. Lett., Vol. 37, pp. 295-298 (August, 1980).
40. C. J. Doherty, T. E. Seidel, H. J. Leamy, and G. K. Celler, "Formation of p-n Junctions and Ohmic Contacts at Laser Processed Pt-Si Surface Layers," J. Appl. Phys., Vol. 51, pp. 2718-2721 (May, 1980).
41. H. W. Curtis, "Integrated Circuit Design, Production, and Packaging for System/38," IBM S/38 Technology Development Report, pub. by GSD Tech. Comm., Atlanta, Georgia (1978).

42. D. D. Tang, T. N. Ning, R. D. Isaac, G. C. Feth, S. K. Wiedmann, and H. N. Yu, "Sub-nanosecond Self-aligned I<sup>2</sup>L/MTL Circuits," IEEE Internat. Electron Device Meeting., Tech. Digest, pp. 201-203, Washington, D.C., (December, 1979), also to be published in IEEE Trans. Electron Dev. (August, 1980).
43. R. A. Gdula, "The Effects of Processing on Radiation Damage in SiO<sub>2</sub>," IEEE Trans. Electron Dev., Vol. ED-26, pp. 644-646 (April, 1979).
44. J. M. Aitken, "1 $\mu$ m MOSFET VLSI Technology: Part VIII -- Radiation Effects," IEEE Trans. Electron Dev., Vol. ED-26, pp. 372-378 (April, 1979).
45. D. J. DiMaria, L. M. Ephrath, and D. R. Young, "Radiation Damage in Silicon Dioxide Films Exposed to Reactive Ion Etching," J. Appl. Phys., Vol. 50, pp. 4015-4021 (June, 1979).

## ELECTRON BEAM TESTING AND RESTRUCTURING OF INTEGRATED CIRCUITS\*

By

D. C. Shaver

Lincoln Laboratory, Massachusetts Institute of Technology  
Lexington, Massachusetts 02173

Dramatic improvements in the cost, performance, and reliability of a digital system can be obtained if the system is integrated on a single chip. Many systems are sufficiently complex that the die size resulting from integration would be very large with a low probability of producing a perfect, functioning die. Since there is a real need for larger integrated systems than can be fabricated free of defects, it is likely that techniques which can locate and "wire-around" defects will be useful and will allow the die-size to increase, perhaps to full-wafer size.

A plausible scheme for fabricating a large system is:

- (i) Design the large-scale system in a highly modular fashion. Partitioning into subsystems should stress minimum interconnection requirements between the subsystems, complete testability of each subsystem, and minimum number of subsystem types. This last requirement suggests that one should try to design and fabricate a single subsystem type, and that each subsystem would be assigned unique functions by a customizing operation. Finite state machines containing PLAs or ROMs are examples of subsystems which could be easily customized.
- (ii) Customize and test the individual subsystems. If a particular subsystem does not function properly, a spare one would be customized and tested.
- (iii) Interconnect working subsystems to form the large-scale system.

To construct such a system a flexible tool is required to allow subsystem customization, testing, and interconnection. The objective of this paper is to demonstrate that a scanned electron-beam provides this flexible tool. Specifically, the electron beam can be used for three essential functions:

- (1) Input injection: the electron beam can be used to apply inputs and to alter the logical state of a subsystem under test.
- (2) Output sensing: the electron beam can be used to sense the presence of a "zero" or "one" state at any one of a large number of test points.

---

\*This work was supported by the Department of the Air Force and the Defense Advanced Research Projects Agency.

- (3) Non-volatile restructuring: the electron beam can be used to open or close switches in the subsystems in a non-volatile manner. The opening and closing of switches is used to provide customization within the individual subsystems as well as control of the discretionary interconnect between working subsystems.

A variety of physical effects including charging and discharging of surfaces<sup>1</sup>, voltage contrast<sup>2</sup>, electron-beam induced current (EBIC)<sup>3</sup>, melting or vaporization<sup>4</sup>, eutectic-formation<sup>5</sup>, threshold shifts in MOSFETs<sup>6</sup>, and decomposition of organometallic vapors<sup>7</sup> may occur when an electron beam interacts with matter, and these effects could be exploited in an electron-beam testing and restructuring tool. Earlier work on electron beam testing of integrated circuits has centered on voltage contrast examination of chips for failure analysis, or stroboscopic measurement of waveforms. Generally a raster scan of the electron-beam is generated and an image of the chip is displayed or, alternatively, a single point is probed and a waveform is viewed on a CRT. The emphasis of the work presented here is to develop methods for computer-controlled electron beam testing of wafer-scale circuits, including restructuring techniques. This emphasis has the following implications:

- (1) Input injection, output sensing, and programming of non-volatile switches must all be achievable in a single system.
- (2) The test point to be probed or switching element to be programmed is selected by a computer-controlled deflection of the beam to the appropriate coordinate. The beam is unblanked only over selected points, so the entire circuit is not exposed to the beam.
- (3) Testing can be fully automatic since a mask level description provides coordinate information for the electron-beam system. Potentially, node extraction and switch level simulation<sup>8</sup> can be used to generate test sequences automatically from the mask description.
- (4) The objective is to perform only functional testing (i.e., checking for logical ones or zeroes), not parametric measurements.
- (5) The integrated circuits must be designed to be compatible with electron beam testing. Specific structures incorporated in the mask-level specification make the electron-beam testing and restructuring possible. A wafer-scale power grid is used to supply power during electron beam testing.

Figure 1 illustrates a possible layout for a wafer-scale electron-beam testable system. The central portion of the wafer containing active subsystems and discretionary interconnect would be probed only by the electron beam. Some large test and power pads at the wafer perimeter could be contacted by relatively large probes in the cassette which holds the wafer in

105852 - N

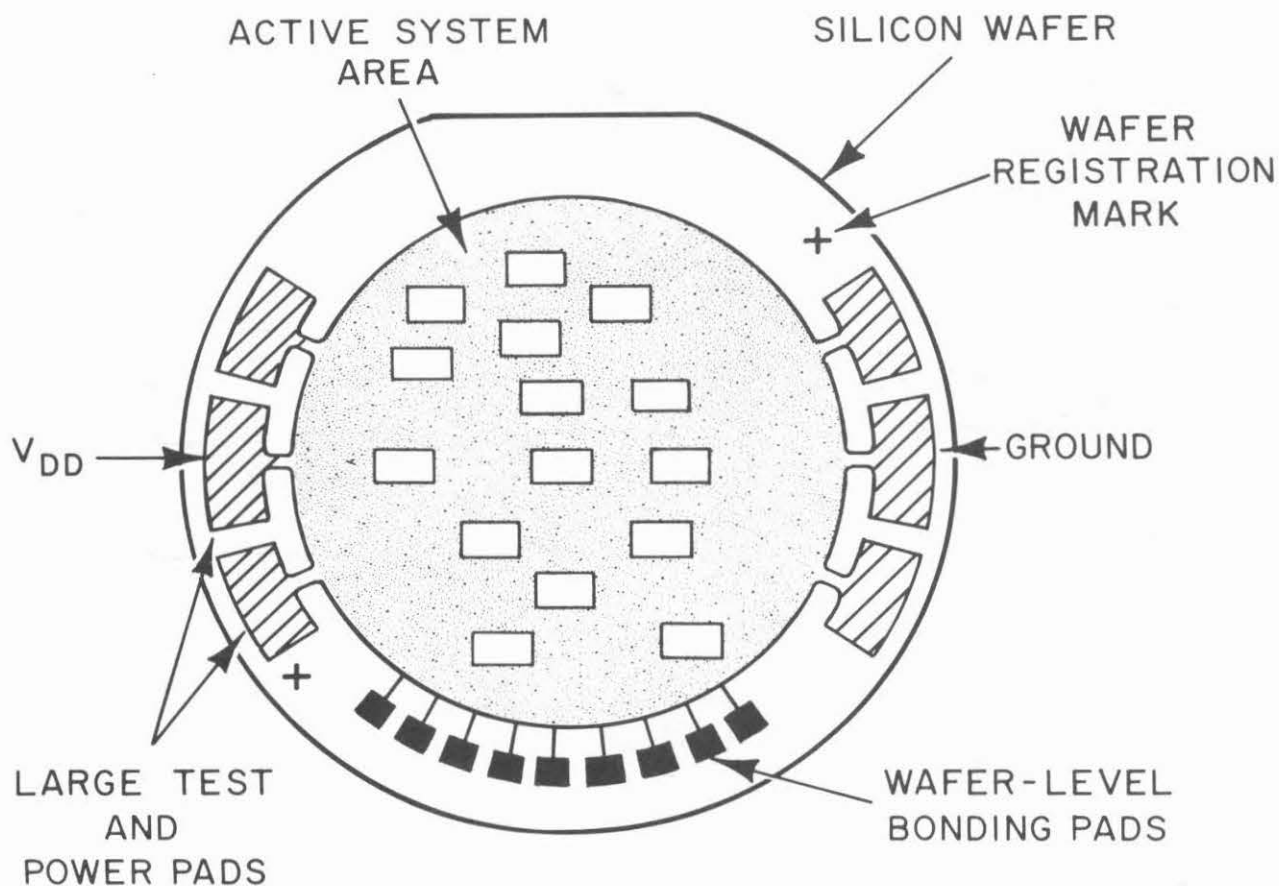


FIGURE 1: A possible layout for a wafer-scale system designed for electron beam testing and restructuring.

electron beam system. Only a very limited number of these pads would be required since the electron beam multiplexes use of the pads. A set of wafer-level bonding pads would be used to connect the wafer-scale system inputs and outputs in the final package. Figure 2 illustrates schematically a possible arrangement within the active subsystem area.

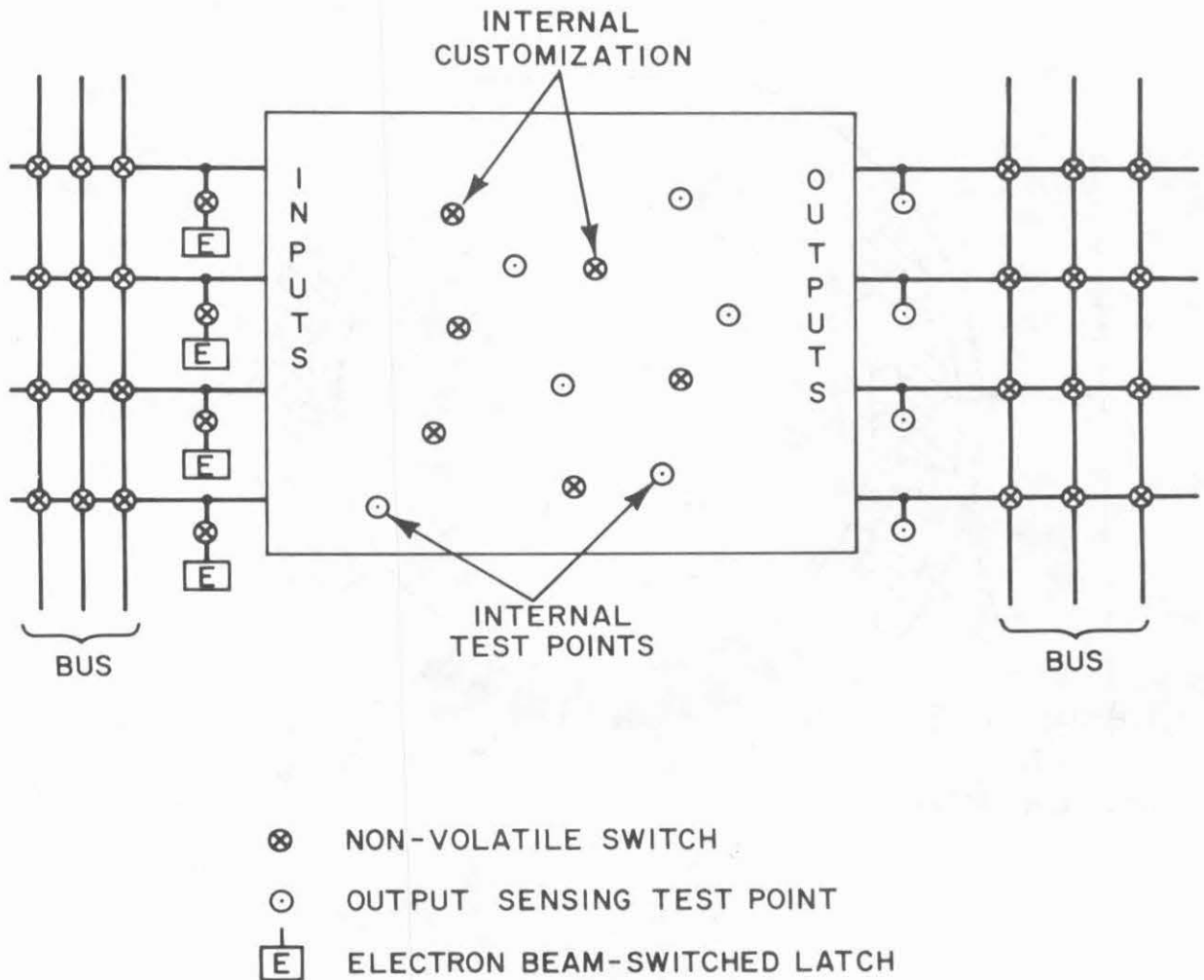


FIGURE 2: A schematic representation of a subsystem designed for electron beam testing and restructuring. ESLs can be used to apply inputs to the subsystem and sensing points are provided within and at the outputs of the subsystem. Non-volatile switches provide subsystem customization as well as flexible interconnect.



### LOGICAL "OUTPUT" SENSING TECHNIQUES

Only specific techniques which are useable with n-channel MOS (NMOS) technology and which require no significant modification of our commercially available ETEC LEBES-D electron beam lithography system will be described. Since the system is also routinely used for mask-making and direct-write lithography, techniques were chosen to be compatible with operation of the machine in a lithography mode. Specifically, the beam-defining aperture size and secondary-electron detector placement were not optimum, and techniques such as organometallic decomposition which would cause system contamination were avoided. The most obvious technique for sensing the presence of a logical "zero" or "one" is to utilize voltage-contrast in the secondary electron signal. In particular, if the incident electron beam is directed at a flat aluminum test pad a few microns in diameter, and if the test pad is surrounded by a grounded metal ring several-microns wide, a strong modulation of the secondary electron signal will be obtained as the potential at the test pad is varied from zero volts to a +5 V logic level. With the test pad at +5 V, the lowest energy secondaries will not escape from the vicinity of the pad, and the secondary signal will be reduced. Under ideal conditions, a very strong modulation of the secondary electron signal can be achieved and virtually all secondary electrons leaving the pad can be collected. In this ideal case, a high enough signal to noise ratio can be obtained in the secondary signal (with modest incident beam currents) to allow reliable discrimination between a "one" and "zero" logic state with a beam unblank time of less than one microsecond. Thus, in principle, the beam could be deflected to examine more than one million test points per second. Put another way, if a chip were clocked at a 10 kHz rate, 100 selected internal test points could be examined during each clock cycle.

This voltage contrast functional probing should provide excellent testability with minimal area overhead for test points. Unfortunately, results on the LEBES system have been disappointing. Measurements indicate that the secondary electron detector in this system is very poorly placed and receives less than one electron for every ten thousand electrons incident on the sample. For comparison, with a good detector arrangement one could receive about 1 electron for every 10 incident electrons under comparable bombardment conditions. In addition, most of the low-energy electrons collected by the detector appear to be produced in the vicinity of the detector by high-energy electrons backscattered from the specimen which results in a very poor voltage contrast modulation of the "secondary" signal. These two effects reduce the logic level measurement rate to only about 100 Hz. It is anticipated that detector improvements will improve this rate by several orders of magnitude.

A second logic-level sensing technique eliminates most of the difficulties encountered with voltage contrast probing. This technique is capable of logic level sensing at rates of at least 1 MHz, and is free of contamination, charging, and crosstalk effects encountered with voltage contrast techniques. The incident electron probe is pointed at a specially designed test point and is used to select that test point for examination. When an

electron beam penetrates a semiconductor, electron-hole pairs are generated. These electrons or holes can be collected by a p-n junction, causing a current to flow in the integrated circuit. Reasonable incident beam currents might be as large as a few hundred nanoamperes, but such low currents will not appreciably perturb static circuits. However, if an incident electron has a 5-20 keV energy it can generate about 1000-4000 electron-hole pairs. If these are collected by a junction, currents of several hundred microamperes can be induced in the integrated circuit which is larger than the current which is usually supplied by a depletion-mode pullup. In NMOS, since p-n junctions are formed between all  $n^+$  diffused conductors and the substrate, the e-beam can be used to pulldown any diffused conductor to substrate potential, which is usually ground. Thus, by designing so that selected diffused conductors are accessible to the electron beam, it is possible to produce a low logic level at a selected point merely by pointing the electron beam and unblanking it.

This forms the basis for a number of possible sensing schemes including the electron beam controlled multiplexor shown in Fig. 3. In this scheme each of a number of test points (shown as A, B, C...) is connected to the gate of a FET added specifically for test purposes. The drains of these FET's are connected to a test bus which can be the power bus, or a special test bus can be used to reduce the capacitive loading and improve sensing speed. The source terminals of the FET's are left disconnected, and the source-substrate diode provides the electron-beam probed point. If, for example, test point B is electron bombarded, the diode will be driven towards forward bias. Depending on the state of test point B, the FET switch will be closed or open and the bombarded diode will be able or unable, respectively, to draw current from the test bus. Thus, the appearance of current on the test bus synchronous with the bombardment of the test point will indicate the logical state. A single wire provides access to many test points with the electron beam providing test-point selection. A number of variations on this basic theme are possible including tree-structured buffered busses. The operation of a single test point will be demonstrated later in this paper.

#### INPUT INJECTION

Hole-electron pair injection by electron beam provides the basis for applying inputs to a powered-up device without actual input connections. One device, the electron-beam switched latch (ESL), provides a means for applying static inputs to a system under electron beam control. The electron beam is used to control a set-reset flip-flop by bombarding either a set-to-one or a set-to-zero control diode. ESLs can be used to provide stable inputs or clocks, or as volatile control for programmable links during testing.

105855-N

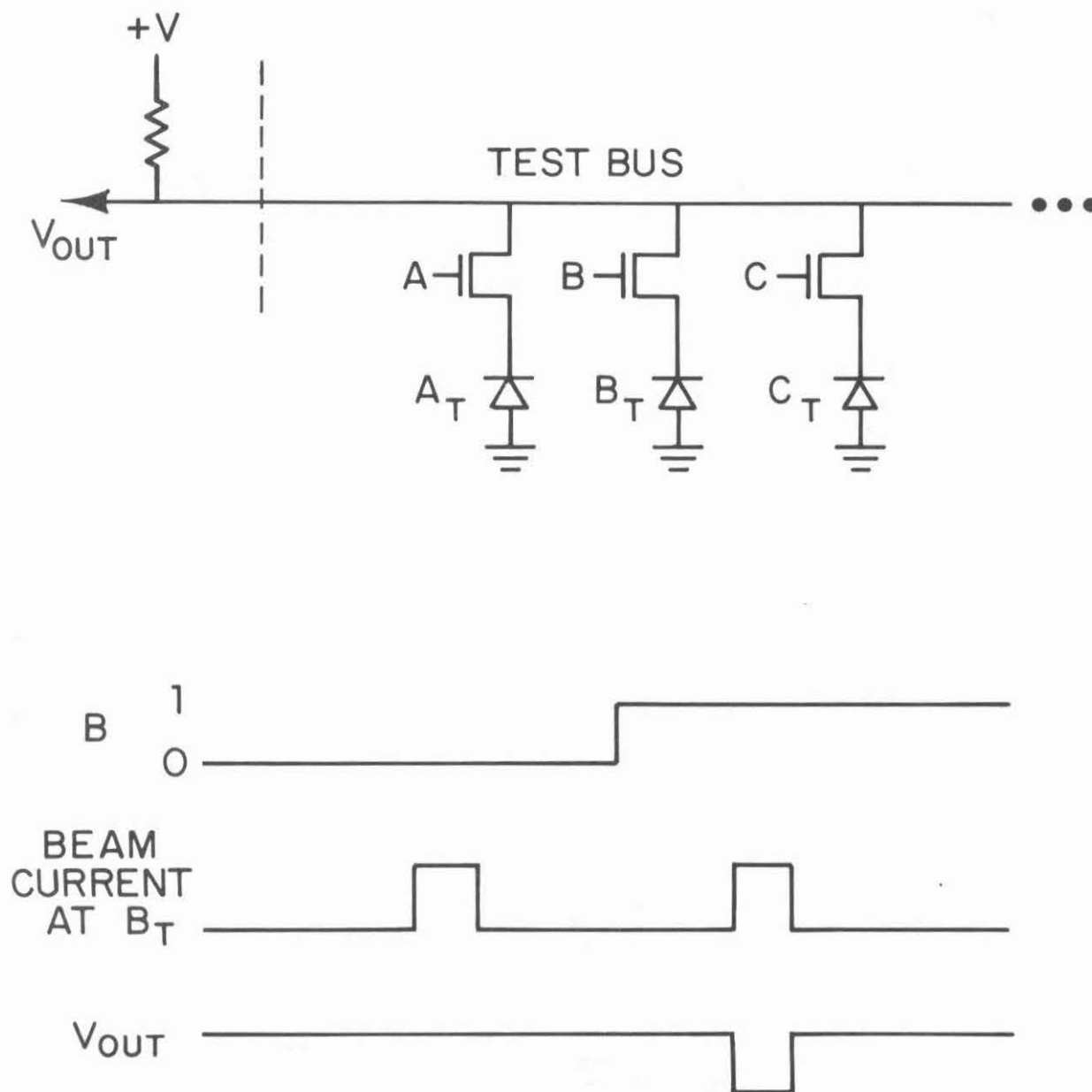


FIGURE 3: A simple electron-beam controlled multiplexor for sensing a selected test point.

Figure 4 is a schematic representation of an electron-beam switched latch. The latch is a pair of cross-coupled inverters with pulldown access provided for the electron beam. Figure 5 shows an actual layout of an NMOS ESL. Electrically-long pullups are used to reduce the required pulldown current to less than 10  $\mu$ A, and the e-beam pulldown points have been extended well away from the active transistors. Uncovered contact cuts provide electron beam access. This relatively large layout provided many advantages during preliminary experiments, but the ESL could be made as small as two minimum size inverters. Two additional structures are attached to the output of the ESL in Fig. 5. One is an inverter which was added to make the power supply current unequal for the two states of the ESL which made it possible to test ESL operation by simply monitoring supply current. The second structure is a one-bit slice of the electron-beam controlled multiplexor described in the previous section. Thus, this simple chip can provide a feasibility demonstration of combined electron-beam input injection and output sensing.

A chip fabricated at Hewlett-Packard as part of the MPC-580 multiproject chip run was wire-bonded and placed in our ETEC electron beam lithography system. Only two wires, supplying power and ground, are attached to the chip. After setting the beam parameters (5 kV, 7nA) and registering the chip to the electron-beam coordinate system, electron-beam control of the test chip began. As shown in Fig. 6, successful electron-beam switching of the latch and electron-beam probing was achieved. The lower trace in Fig. 6 shows the electron beam x-deflection signal and indicates at which of three locations the beam is positioned. An upwards deflection on this trace corresponds to a leftward motion (in Fig. 5) of the beam. The beam cycles among three positions. From highest to lowest on the trace the positions are set-to-zero, set-to-one, and sense-output. The top trace in Fig. 6 shows turn-on (unblanking) of the electron beam as a small downward blip. The center trace shows an AC-coupled record of the supply current monitored across a small sense resistor. Moving from left to right across Fig. 6, the beam is initially positioned at the set-to-zero location. When the beam is unblanked, the latch switches to zero causing the downward transient in supply current since the latch was designed to draw less current in the zero state. Then, the beam position shifts to the sense location and is again unblanked; no sense pulse appears on the supply current, so the latch is zero. Then, the beam moves to the set-to-one position and unblanks. The immediate change in supply current indicates the ESL has changed state. Finally, the beam is moved to the sense position again and a small pulse appears in the supply current synchronous with the unblanking of the beam. The output is a one. Figure 6 demonstrates complete electron beam control and probing of a small logic circuit, and the principle is extendable to complex systems. The ESL described above could be switched with a 380 ns pulse.

The ESL provides electron-beam control of static logic levels. The electron beam may be used to great advantage in dynamic circuits as shown in Fig. 7. Many NMOS integrated circuits are designed as two-phase clocked, dynamic finite-state machines. A very large number of electron-beam

105854-N

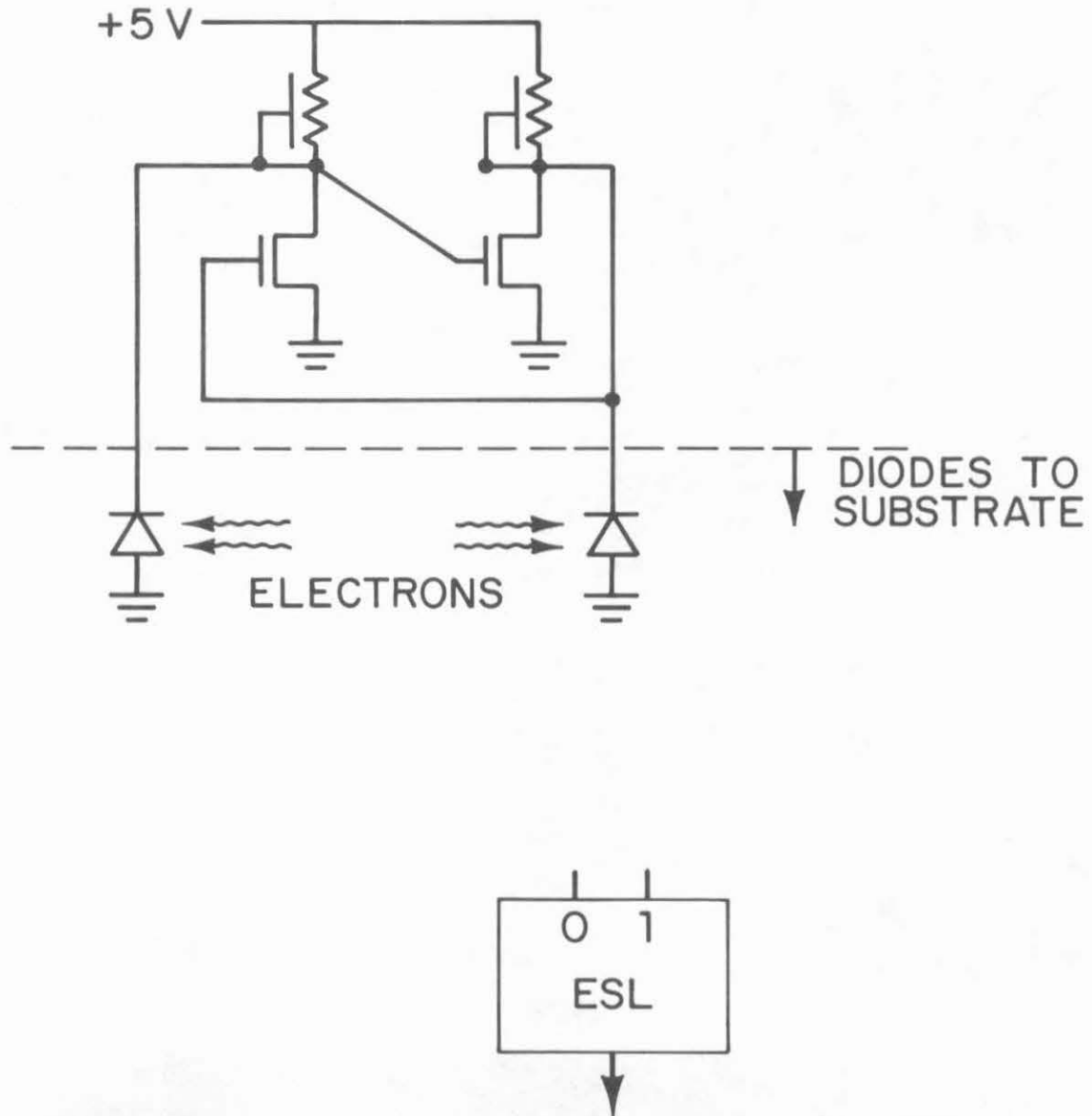


FIGURE 4: Schematic representation of an electron beam switched latch.

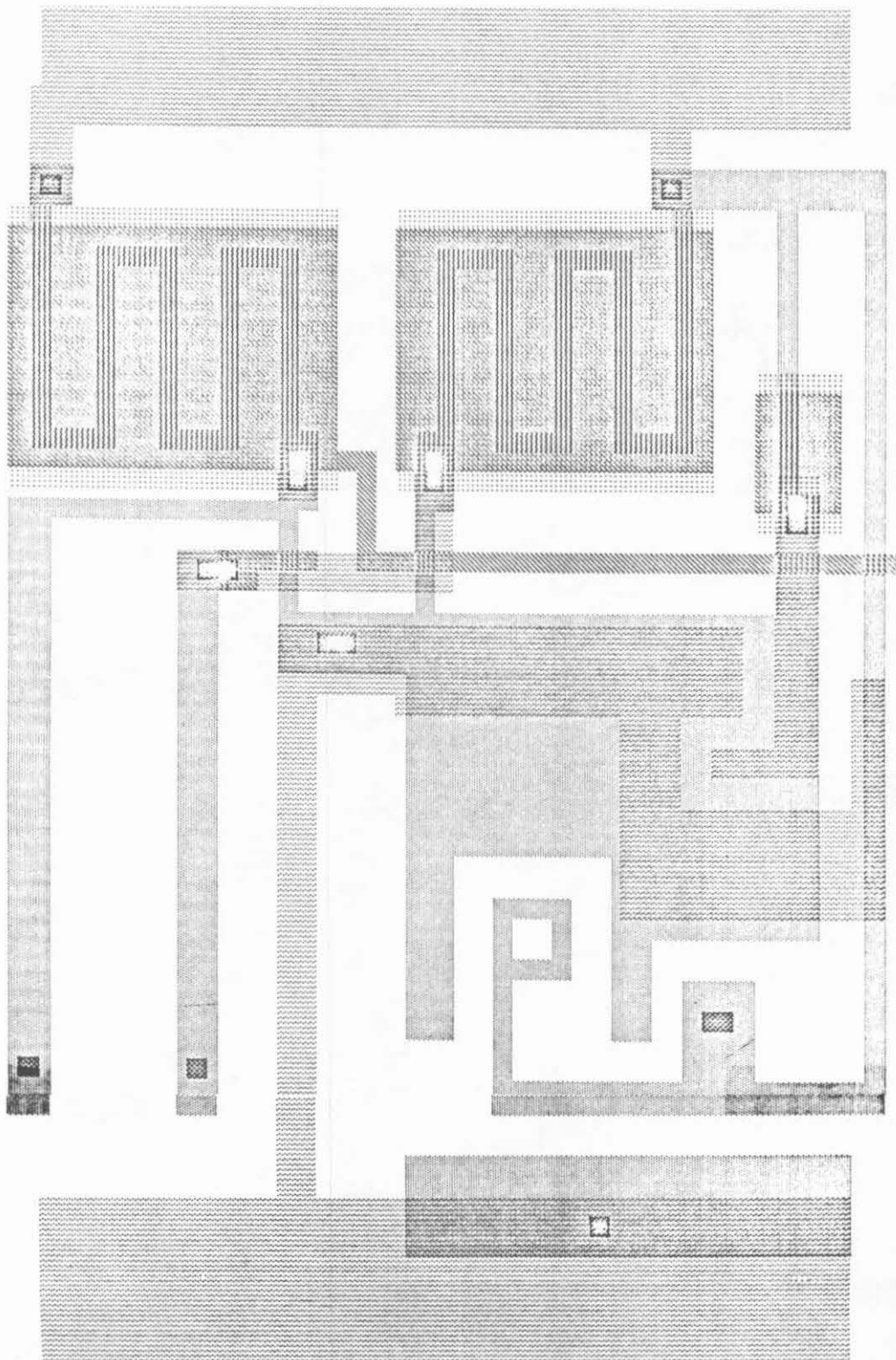


FIGURE 5: NMOS layout of an electron-beam switched latch. Top pad is +5 V and bottom pad is ground. The three dark contact cuts above the ground pad correspond, from left to right, to "set-to-zero", "set-to-one", and "sense" points for the electron beam.

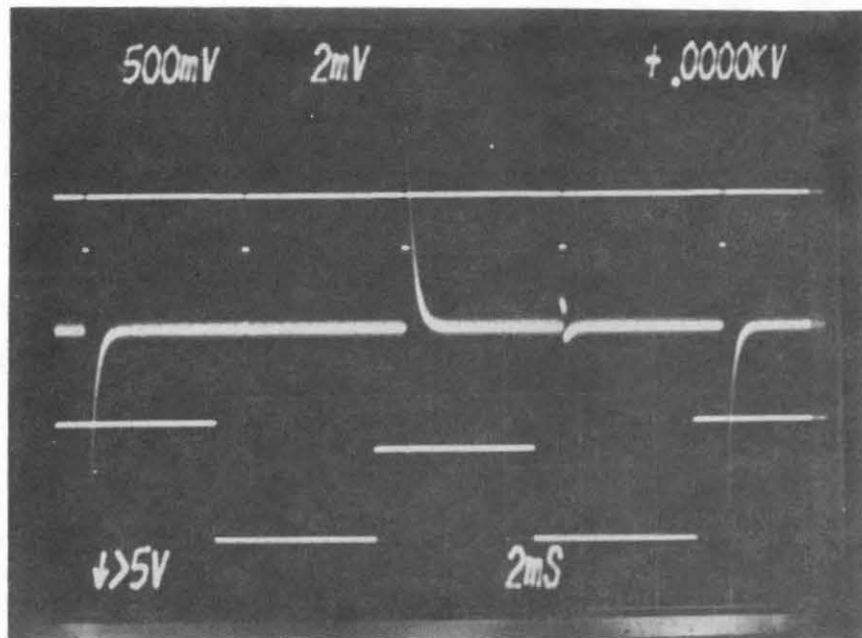


FIGURE 6: An electron beam switched latch in operation. Downward pulses on top trace show beam unblanking. Center trace is AC-coupled supply current to ESL. Lower trace shows beam deflection.

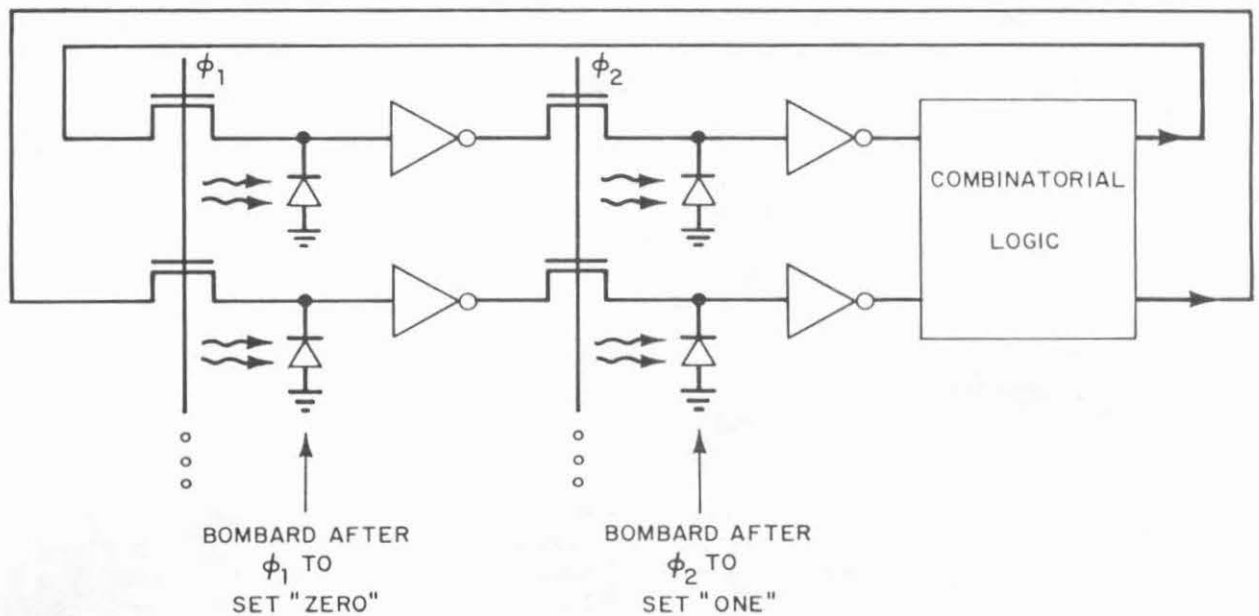


FIGURE 7: A dynamic finite-state machine showing how the electron beam is used to alter the internal state.



accessible test points are available, and the beam is used to selectively dump stored charge after the clock-controlled pass transistors are cut-off. The ability to alter sequentially internal states in the finite-state machine and to probe the consequences will allow an unprecedented degree of testability.

#### ELECTRON BEAM CONTROL OF NON-VOLATILE SWITCHES

The ability to open and close selected switches on the wafer is essential if one wishes to customize subsystems and modify interconnects. Ideally these switches should be easily flipped from on to off (and vice-versa) under e-beam control and should exhibit high off resistance and low on resistance. Once programmed by the beam they should retain their state indefinitely. Two types of electron-beam alterable switches, field-oxide FETs and floating-gate FETs, are described below.

The field oxide FETs are parasitic devices present in all NMOS processes. When a polysilicon or metal conductor runs over thick field oxide, a parasitic FET is formed between adjacent diffused conductors. Normally, this FET is off since the combination of thick oxide and field implantation raises its threshold to many volts. Electron-beam irradiation of oxides with a bias applied across the oxide can result in a large buildup of positive charge in the oxide, presumably due to hole-trapping. Specifically, if a positive voltage is applied across thick oxide and the FET is irradiated a thin layer of positive charge will accumulate near the silicon surface producing an effect equivalent to applying the bias across a thin oxide of only about a hundred angstroms thickness. Since this charge remains trapped in the oxide after the irradiation and bias are removed, a large ( $> 50$  V) negative threshold shift is induced in the FET. For example, a polysilicon gate field-oxide FET formed in the MPC-580 run exhibited a threshold of  $> 10$  V. Thus, for normal logic levels between 0-5 V, the gate cannot turn-on the FET, and adjacent diffused conductors are not connected. This FET was strongly turned on by applying +3 V to the gate and irradiating the gate oxide. As shown in the top of Fig. 8, the FET is strongly turned on after e-beam irradiation with a positive gate-programming voltage. The four closely spaced characteristics correspond to gate voltages of 0, 1, 2 and 3 V indicating that the FET is strongly on regardless of the gate voltage after programming. By biasing the gate at zero volts and irradiating again, the FET can be turned off. The lower half of Fig. 8 shows the drain-source characteristic of the FET after this operation. For gate voltages varying from 0-5 V, the FET is off.

E-beam programmed field-oxide FETs provide a simple, reprogrammable element for restructuring and customization. The voltage applied to the programming gates during electron beam bombardment can be generated locally from the output of an ESL or can be supplied over a single programming wire such as the power bus. The retention time of the on state is at least a few weeks for devices operated at room temperature, but the retention characteristics are likely to be poor if the devices are operated at  $150^{\circ}\text{C}$  for even short periods. Nonetheless, this e-beam controlled switch should be very



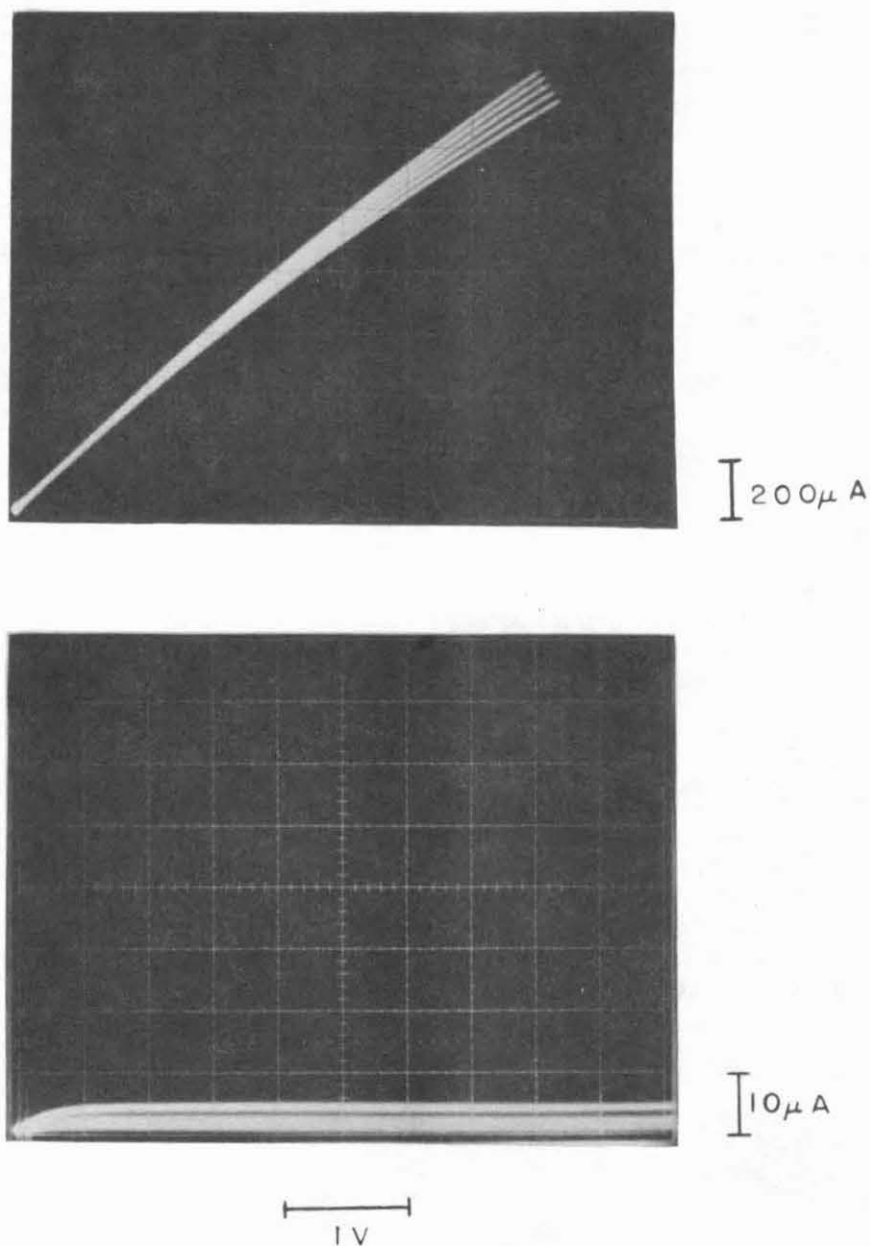


FIGURE 8: A field-oxide FET non-volatile switch after e-beam programming. Switch is initially off. Top picture shows FET characteristic after e-beam programming to the "on" state. Note the low sensitivity of the FET conductance as the gate voltage is varied from zero to +3 volts. Bottom picture shows "off" state after e-beam reset.

useful if the devices are operated at liquid nitrogen temperatures, and will certainly prove to be a useful element in laboratory studies of large-scale restructurable systems. Less than  $1 \mu\text{C}/\text{cm}^2$  of 20 kV bombardment is required to produce the threshold shifts, and a programming time of less than 1  $\mu\text{s}$  per switch is possible.

Electron beam programmed floating gate FETs should provide a reliable, truly non-volatile switch. Electron storage on floating gate FETs is the basis for information retention in UV-erasable EPROMs which have enjoyed wide commercial acceptance and have excellent retention characteristics. FAMOS<sup>9</sup> electrically-programmable ROMs are programmed by hot-electron injection through the thin gate oxide where the hot electrons are generated by avalanche breakdown of the drain-substrate junction. Electron-beam programming of floating gate devices is possible if an electron beam is directed at an oxide-covered gate, causing electrons to penetrate the oxide and become trapped on the gate. The trapped electrons will charge the gate to a negative potential.

Some preliminary experiments have been performed on depletion-mode floating gate devices fabricated as part of MPC-580. These devices were normally on, and could be turned off by electron bombardment of the gate. Figure 9 shows a drain-source characteristic for a FET before (top photo) and after (bottom photo) electron-beam programming. At a 5 V drain-source bias, the current is 100 times greater in the on state than in the off state. The residual current of 5  $\mu\text{A}$  in the off device was determined to be the result of formation of a weak buried channel in the depletion mode device. Biasing the substrate to -0.4 V increased the on : off ratio to better than 10000 : 1. A slight modification of the depletion mode implant would greatly reduce the magnitude of the buried channel. The large floating gate devices used for these experiments required 370  $\mu\text{s}$  to insure complete turn off using a modest 2 nA beam. By using larger beam currents and smaller devices, programming times of a few microseconds could be achieved.

## CONCLUSION

Control of the state of a simple NMOS integrated circuit using an electron beam has been demonstrated. Techniques have been demonstrated for input injection, output sensing, and programming of non-volatile switches using a commercially-available electron beam lithography system and conventional NMOS technology. Many of the techniques for input injection and output sensing have also been demonstrated using light beams, but the experimental apparatus was very limited. Though many approaches to "scanning-beam" probing and restructuring are feasible, the state-of-the-art in electron beam deflection, blanking, etc., is highly advanced, and electron beam probing and restructuring could be a reality in the very short term. This can provide us with a flexible tool to understand the problems in constructing wafer-scale systems.

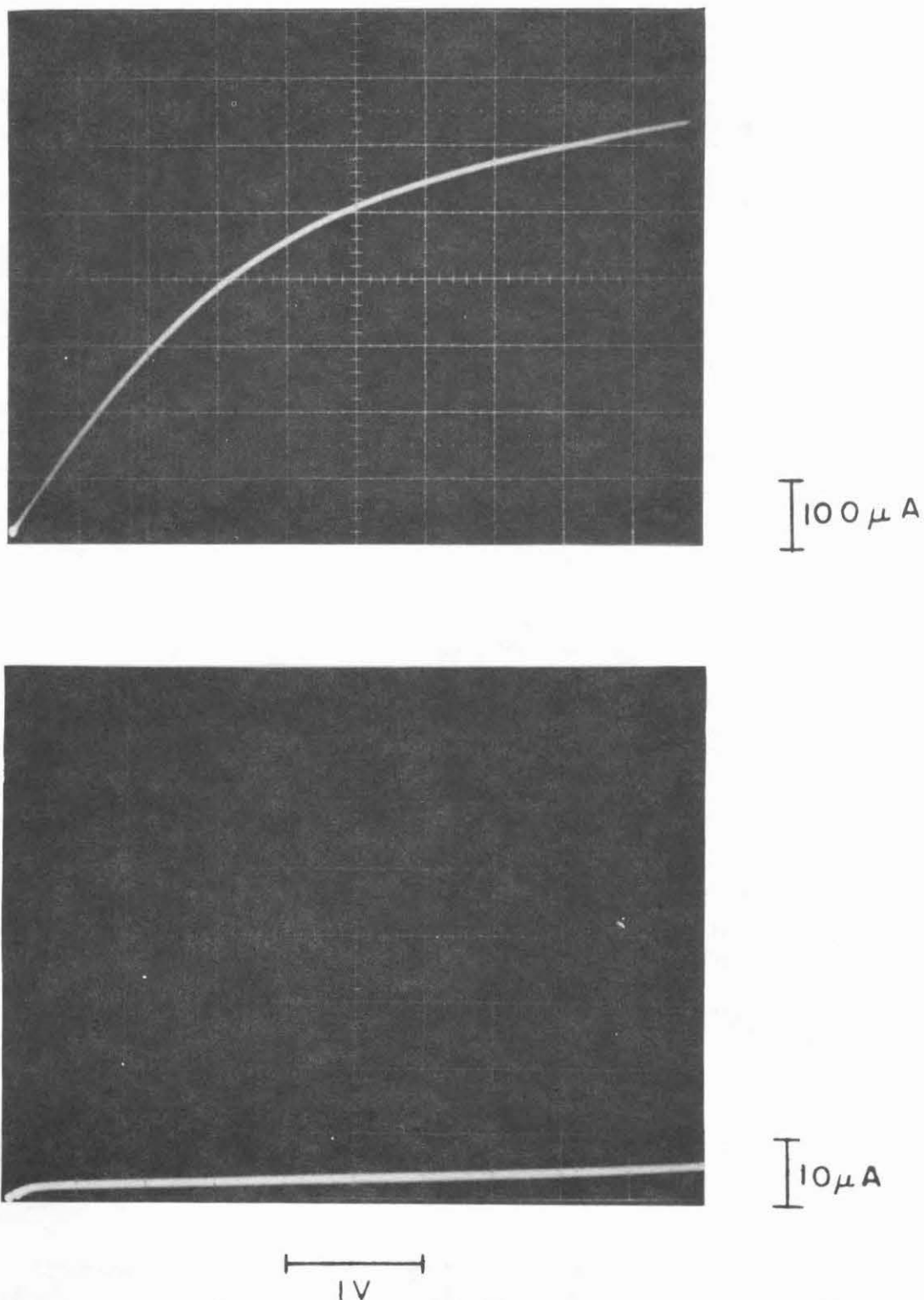


FIGURE 9: A depletion-mode floating gate FET before (top photo) and after (bottom photo) electron beam programming.

# ACKNOWLEDGMENTS

The test chips used for these experiments were fabricated as part of the DARPA sponsored MPC-580 multiproject chip. I would like to thank everyone at Hewlett-Packard, Micro Mask, and Xerox who made this multiproject chip a success. At Lincoln Laboratory, Dr. B. Burke provided test FETs used for initial experiments, and P. Daniels and D. Klays provided assistance with packaging and bonding.

# REFERENCES

1. P. E. Kudirka and C. K. Crawford, "Potential measurement and stabilization of an isolated target using electron beams", Solid State Electronics, 15: 987-992 (1972).
2. Eckhard Wolfgang, Rudolf Lindner, Peter Fazekas, and Hans-Peter Feuerbaum, "Electron-beam testing of VLSI circuits", IEEE Trans. Electron Devices, ED26: 549-559, April 1979.
3. James R. Beall and Leon Hamiter, Jr., "EBIC - A valuable tool for semiconductor evaluation and failure analysis", p. 61-69, IEEE Reliability Physics Symposium, 1977.
4. J. Vine and P. A. Einstein, "Heating effect of an electron beam impinging on a solid surface, allowing for penetration", Proc. IEEE, 111: 921-930 (1964).
5. C. G. Kirkpatrick, J. F. Norton, H. G. Parks, and G. E. Possin, "New concepts for electron-ion beam and electron-electron beam memories", J. Vac. Sci. Technol., 15: 841-844 (1978).
6. N. C. MacDonald and Thomas E. Everhart, "An electron beam activated switch and associated memory", Proc. IEEE, 56: 158-166, February 1968.
7. Allen G. Baker and William C. Morris, "Deposition of metallic films by electron impact decomposition of organometallic vapors", Rev. Sci. Instrum., 32: 458 (1961).
8. Clark M. Baker and Chris Terman, "Tools for verifying integrated circuit designs", Lambda, Volume 1, No. 3, p. 22 (1980); see also Randal E. Bryant, "An algorithm for MOS logic simulation", Lambda, Volume 1, No. 3, p. 46 (1980).
9. Dov Frohman-Bentchkowsky, "A fully-decoded 2048-bit electrically programmable FAMOS read-only memory", IEEE J. of Solid State Circuits, SC6: 301-306, October 1971.

# Two Timing Samplers

Edward H. Frank

Robert F. Sproull

Department of Computer Science  
Carnegie-Mellon University  
Pittsburgh, PA 15213

## Abstract

Testing VLSI chips presents a variety of problems, some of which can be solved by building *on-chip* testing structures. On-chip testing structures can allow a designer to test aspects of a circuit which might be difficult to test even with expensive test equipment and moreover can provide reasonable testing hardware to designers who do not have access to sophisticated off-chip testing equipment.

In this paper we describe a type of on-chip test structure called a *timing sampler* which enables the designer to accurately measure when on-chip signal transitions occur. The timing samplers we present are simple. They have been fabricated as part of a multi-project chip and experimental results show that they are reasonably accurate as well.

Copyright (C) 1981 Edward H. Frank and Robert F. Sproull

This research was sponsored by the Defense Advanced Research Projects Agency (DOD), ARPA Order No. 3597, monitored by the Air Force Avionics Laboratory Under Contract F33615-78-C-1551.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the US Government.

## 1 Introduction

As VLSI chips become increasingly complex designers are discovering that testing and debugging must be considered in the overall chip design. As a result, testing mechanisms such as scan-in/scan-out (often called shift-register latch (SRL) or level-sensitive scan design (LSSD) [Eichelberger 78]) for accessing internal state and on-chip signature analysis [Frohwerk 77] such as BILBO [Koenemann 79] are becoming more common. In this paper we describe another type of on-chip testing mechanism which is useful for measuring some aspects of circuit performance.

When assessing the performance of a circuit, it is often important to measure the time of occurrence of an on-chip signal. For example, to measure the performance of an adder design, the designer wants to know when the carry signal arrives at the end of the carry chain. The simple expedient of connecting the signal to an output pad driver and measuring the arrival time off-chip is inadequate because a long unknown delay is introduced by the pad driver.

An accurate measurement of the time behavior of a signal could be obtained if we could build an on-chip digital oscilloscope, capable of measuring the signal value at all points in time. We present two circuits for making such measurements in a limited way. The first is a simple latch, which records the value of the signal when a timing signal arrives from off-chip. The second, a C-element, can determine the time at which a single transition appears on the signal to be tested. An additional advantage of the second scheme is that it requires exactly one pad to accomplish the test. Note that the time required to drive the input pad used by the timing signal does not significantly effect the results.

## 2 Latch sampler

A latch can be used to sample the value of a signal under the control of an externally-generated timing signal. We fabricate a latch near the place where the signal to be tested is generated (Figure 1). Signal **T** is the one being tested; **L** is a timing signal generated off-chip; and **R** communicates the result of the test for off-chip analysis. This circuit samples the signal **T**, using the latching signal **L** to determine the time at which the sample is to be taken. **L** is normally high; it is lowered at the instant we want to take a sample (Figure 2). The latch is thereafter closed, with a feedback path ensuring static stability. A complete trace of the behavior of **T** is obtained by repeating the experiment many times, varying the time at which **L** is lowered.

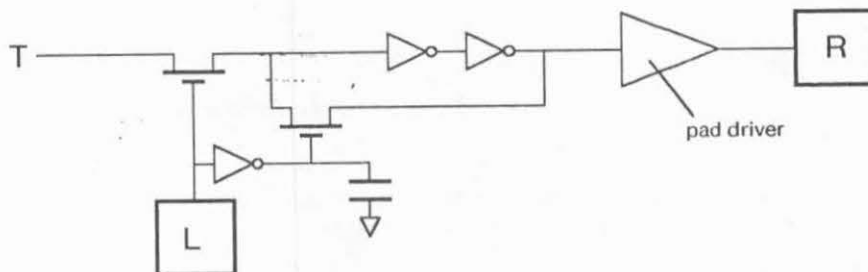


Figure 1: Latch Sampler.

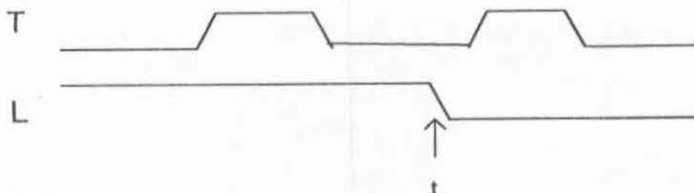


Figure 2: Operating the latch sampler. *t* represents the approximate time of the sample of **T**.

Proper operation of this circuit requires some attention to its design. When **L** is lowered, the pass transistor opens, preventing further change of the charge on the gate of the first inverter of the latch. Note that the gate may be neither fully charged to  $V_{dd}$  nor fully discharged to ground. We now allow the signal to propagate through the latch inverters, and then close the feedback pass transistor. (For this reason, the pullup time of the inverter that inverts **L** to drive the feedback pass transistor gate is arranged to be quite long, substantially longer than the propagation delay through the latch. The pullup time is slowed with a large diffusion-to-substrate capacitor.) After the feedback pass transistor closes, the latch may remain in a metastable state for some time. After allowing sufficient time for metastable exit and delay through the pad driver, we measure the result of the test by sensing pad **R**.

### 3 C-element sampler

The second scheme for measuring timing uses a C-element<sup>1</sup> as shown in the circuit Figure 3. The use of this scheme is illustrated in Figure 4, which shows a test to determine the time at which the signal **T** rises. **L** is initially low, to insure that the C-element is set to zero. **L** is then raised for a while, and lowered at a known time  $t$ . If  $t$  precedes the rise of **T**, the C-element will remain zero; if  $t$  follows the rise of **T**, the C-element will be set to one. The results of the experiment are observed on the output pad **R**. The experiment to detect a falling transition on **T** is the symmetric opposite of the one described above (i.e., **L** is simply the complement of the signal shown in Figure 4.)

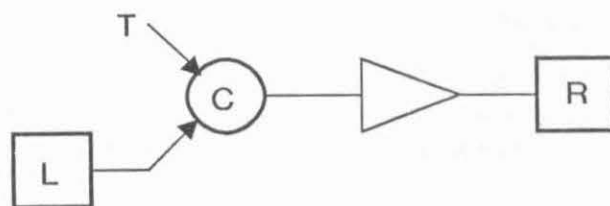


Figure 3: C-element timing sampler.

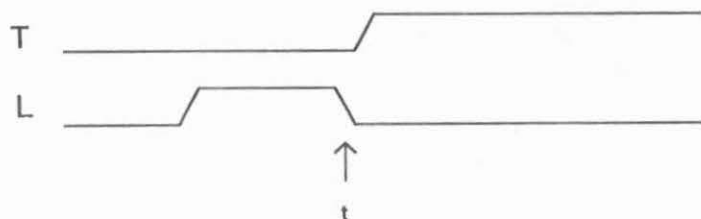


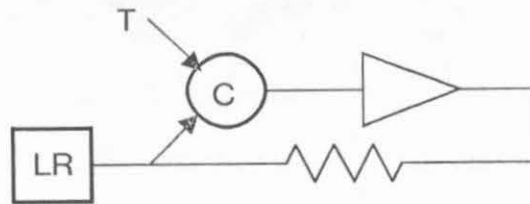
Figure 4: Operating the C-element sampler.

A trick allows us to economize on pads, using a single pad **LR** to communicate both the **L** signal and the results of the test (Figure 5). During the first phase of a test, **LR** is driven from off-chip with the signal shown for **L** in Figure 4. Then the external drive is removed (e.g., by driving **LR** with a tristate driver), and we now detect the results of the experiment by observing whether pad **LR** is high or low. The trick is that although **LR** may change state when the drive is removed, the C-element will not change state. For example, if **LR** is low,

<sup>1</sup>A C-element "... is a bistable device that provides an action similar to hysteresis, in that its output becomes 1 only after *all* of its inputs are 1, and becomes zero only after *all* of its inputs are zero." [Seitz 80]



and the C-element output is high, removing the drive will cause the pad to become high by charging through the resistor, which leaves the C-element high.



**Figure 5:** Using a single pad **LR** for both **L** and **R**.

## 4 Experiments

We have designed and had fabricated a project (as part of two different multi-project chips, MPC580 and M08B) to test these circuits. The circuit diagrams and layouts are shown in Figures 6 and 7. In addition to testing the ideas mentioned in this paper, the project also provides a conventional C-element for various uses (inputs on **T** and **LR**; output on **C**). The circuit diagram of the jig for testing these projects is shown in Figure 8. Basically, we use the variable width pulse generator running at one-half the frequency of the free-running oscillator to vary where the edge of **T** occurs relative to the edge of **L** or **LR**. The **R** and **C** outputs are used to measure the results for the latch and C-element respectively.

As previously mentioned the circuits were fabricated two separate times. Due to a minor mistake<sup>2</sup> the MPC580 version of the chip had a non-functional latch sampler. This error was fixed in the M08B version. The C-element sampler worked in both versions of the chip. We statically tested that the **LR** pin worked as both **L** and **R**. To simplify the test jig, however we, used the **C** output in the dynamic tests.

Figure 9 and Table 1 summarize our results. Although we tested both versions of the C-element, the results were almost identical and so only one set of figures is shown. As can be seen, the latch sampler performed extremely well, with a jitter of at most two nanoseconds. The C-element was not nearly as satisfactory. While the jitter was about the same as the latch, the internal switching delay of the flip-flop caused two problems. First, because of an asymmetry in the C-element, the delay was much greater for the high-to-low C-element transition than for the low-to-high transition. Second, because this delay was much greater than the precision with which we would like to be able to measure signals and moreover because it will vary from fab run to fab run, it makes the C-element as designed here not particularly useful to make accurate timing measurements.

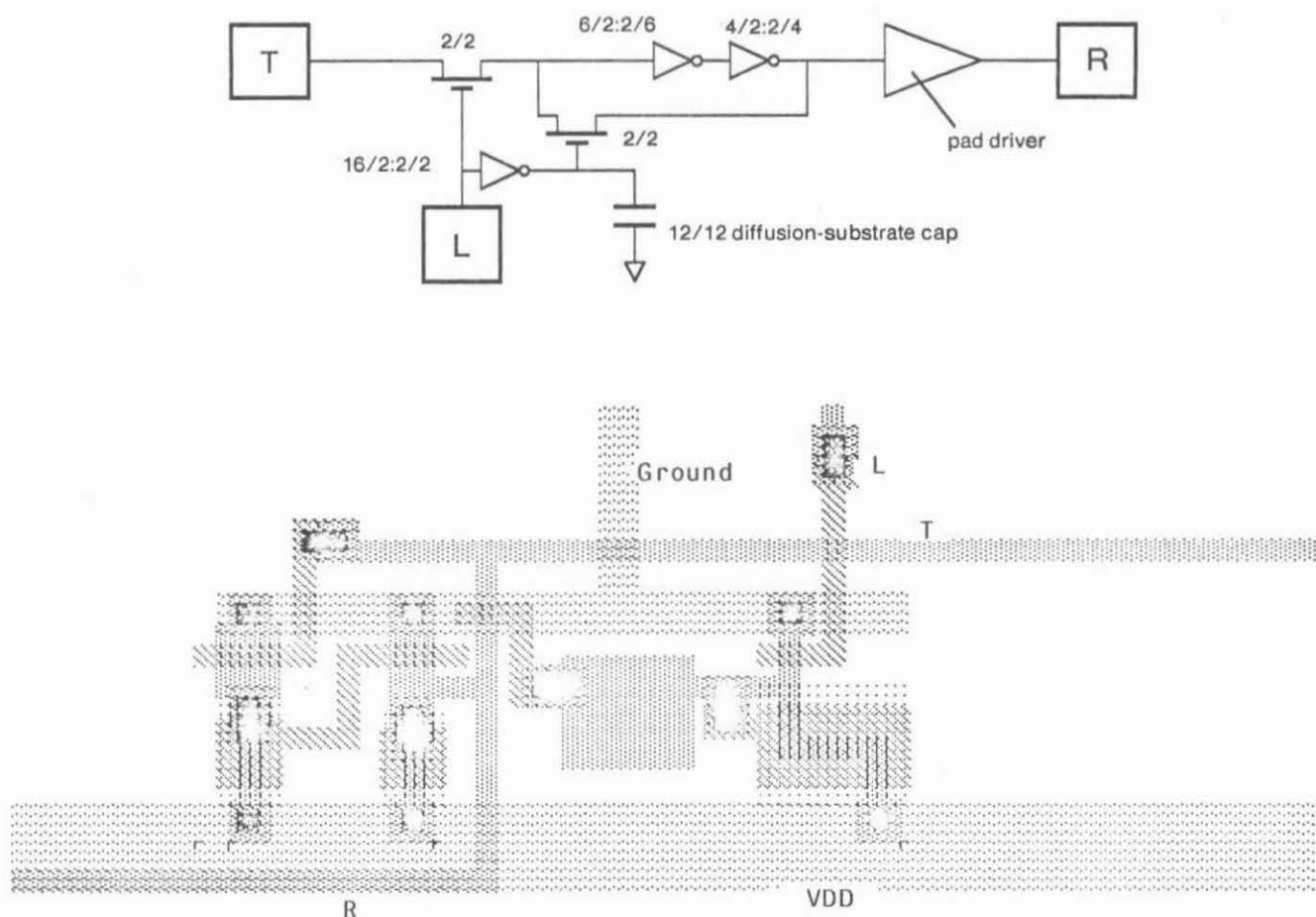
While not shown here, experiments such as measuring the effects of supply voltage and temperature on the timing sampler should also be performed. In addition, Puri [Puri 77] has used a similar form of timing sampler for measuring the minimum pulse width required to disturb a latch.

## 5 Pad Considerations

One objective of any on-chip testing scheme is to minimize the number of pads devoted exclusively to this purpose. The latch scheme seems at first glance to require two pads for each signal, one for **L** and one for **R**. However, a single latching signal **L** could be used to control several latches; this allows us to sample  $n$

<sup>2</sup>Forgetting to wire the **R** pad to ground.





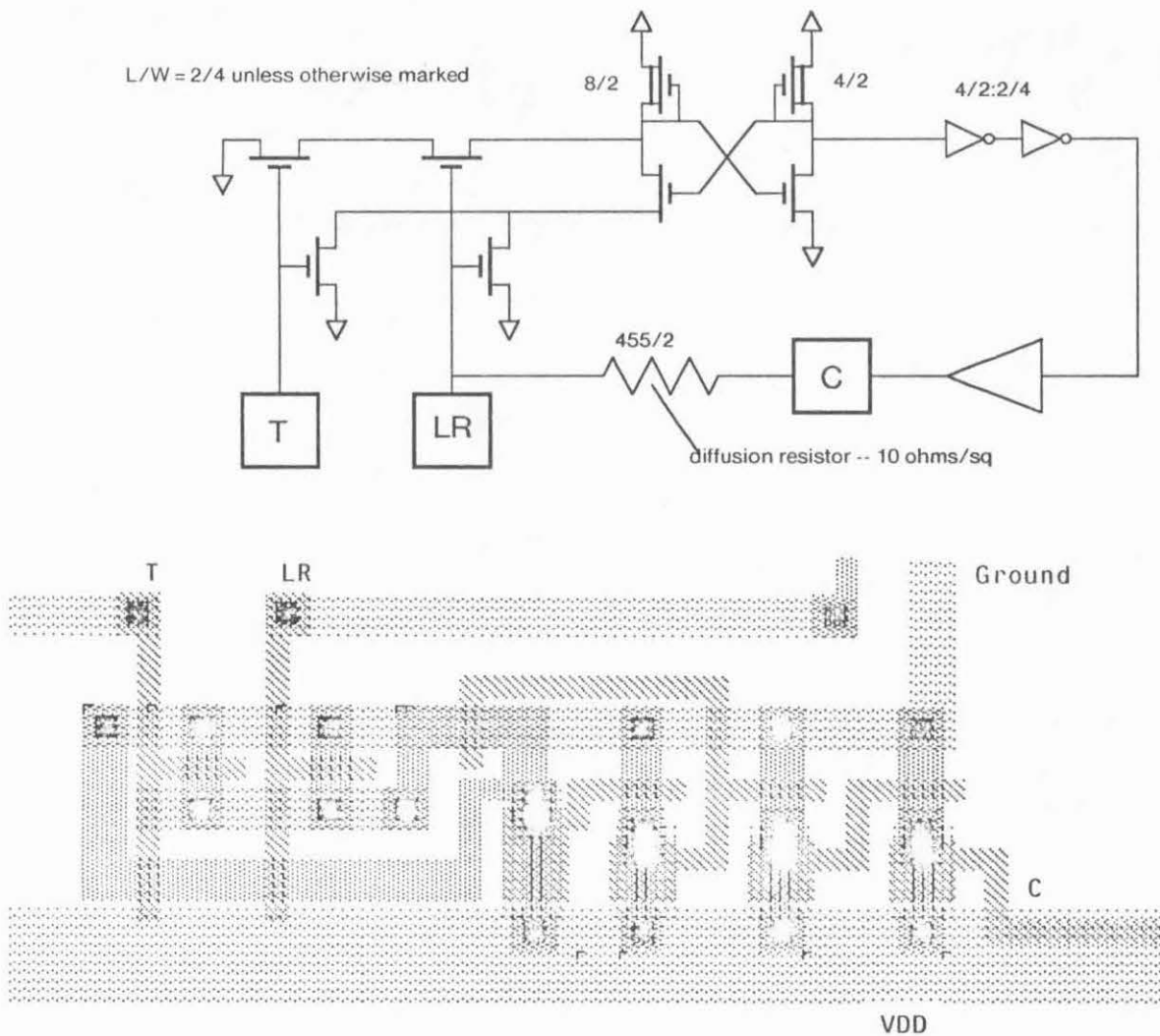
**Figure 6:** Circuit diagram and layout of the latch sampler. Transistor geometries are given as L/W in units of  $\lambda$ .

signals using  $n+1$  pads:  $n$  pads for the results R and a single pad for L. This number can be reduced still further if the chip being tested has incorporated some general provision for scanning out internal state (e.g. shift-register latch). In this case, only a single pad for L is required, because the contents of the various static latches controlled by L can be determined by the scanning mechanism. Moreover, if properly designed, the clock of the scan-out path itself can be used as the L signal.

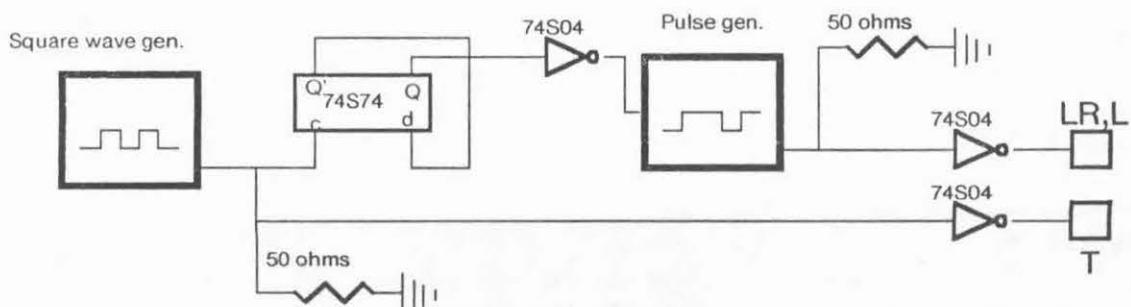
The C-element scheme seems to require a single pad for each signal being tested. However, it too can take advantage of provisions for scanning out internal state: each C-element output can be transmitted off-chip by the scanning mechanism.

## 6 Conclusions

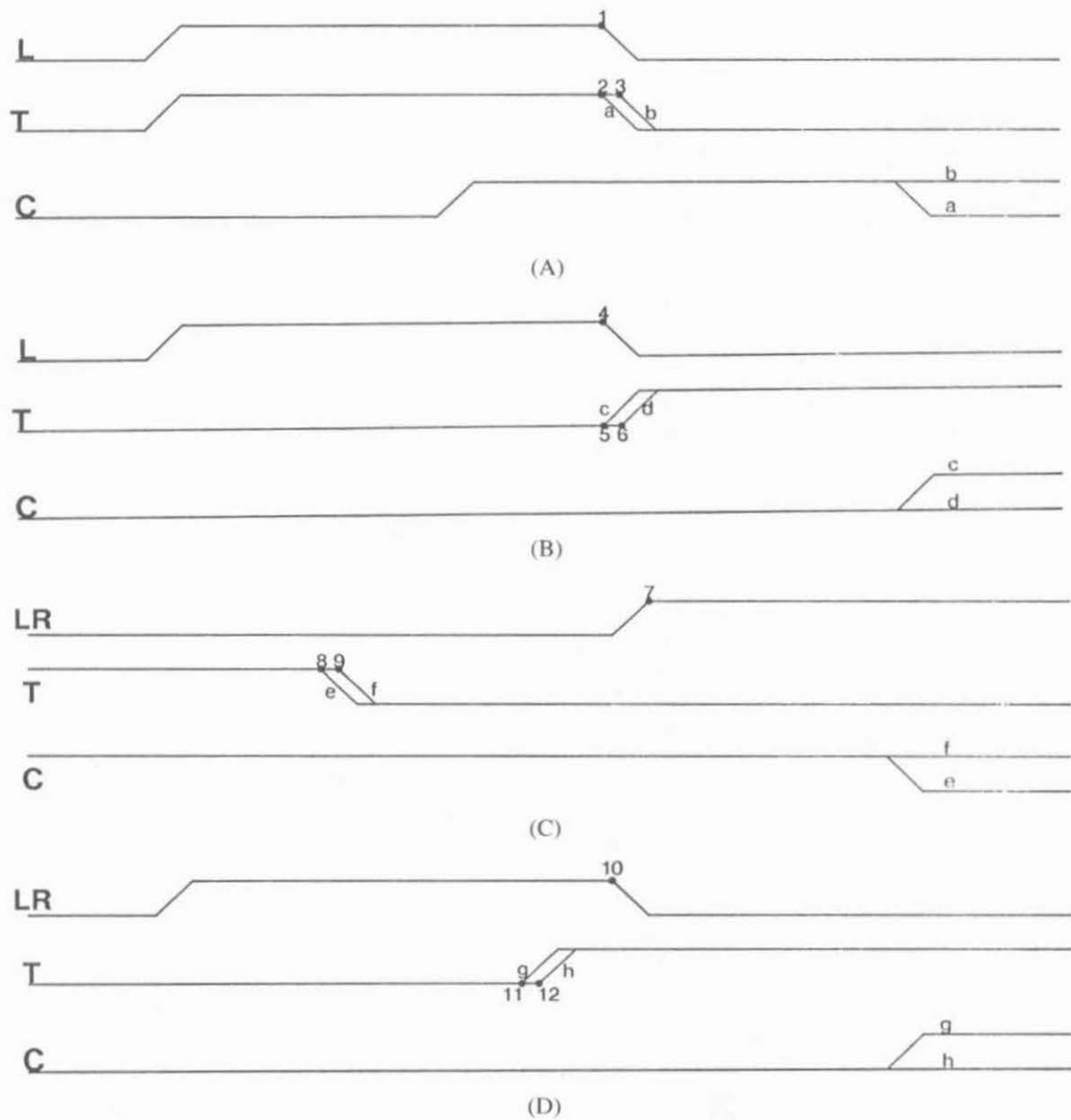
On-chip timing samplers can provide the designer with an effective tool for measuring performance without using sophisticated test equipment. By incorporating these samplers into a scan-out path it is possible for the designer to provide both access to internal state and precise on-chip timing measurements without using many additional pads.



**Figure 7:** Circuit diagram and layout of the C-element sampler. The diffusion resistor is not shown in the layout.



**Figure 8:** Jig used to test latch and C-element samplers.



**Figure 9:** Results of timing experiments. Lower case letters indicate the edge of **T** that generates the corresponding output on **C** or **R**. Numbers and dots are used as reference points for the measurements. The horizontal axis is not to scale. Rise and fall times for **T**, **L** and **LR** are 4ns. (A) Latch sampler used to detect falling edge of **T**. (B) Latch sampler used to detect rising edge of **T**. (C) C-element sampler used to detect falling edge of **T**. (D) C-element used to detect rising edge of **T**.

---

Latch used to detect falling edge of T (Figure 9A):	
T <sub>2</sub> to 1 to produce output a:	0 ns (min)
T <sub>1</sub> to 3 to produce output b:	2 ns (min)
Latch used to detect rising edge of T (Figure 9B):	
T <sub>5</sub> to 4 to produce output c:	0 ns (min)
T <sub>4</sub> to 6 to produce output d:	2 ns (min)
C-element used to detect falling edge of T (Figure 9C):	
T <sub>8</sub> to 7 to produce output e:	32 ns (min)
T <sub>9</sub> to 7 to produce output f:	30 ns (max)
C-element used to detect rising edge of T (Figure 9D):	
T <sub>11</sub> to 10 to produce output g:	12ns (min)
T <sub>12</sub> to 10 to produce output h:	10ns (max)

---

**Table 1:** Timing measurements. All times are +/- 1 ns.

## References

- [Eichelberger 78] Eichelberger, E.B., and Williams, T.W.  
A logic design structure for LSI testability.  
*"Journal of Design Automation and Fault Tolerant Computing"* 2(2):165-178, May, 1978.
- [Frohwerk 77] Frohwerk, R.A.  
Signature analysis: a new digital field service method.  
*Hewlett-Packard Journal* :2-8, May, 1977.
- [Koenemann 79] Koenemann, B., Mucha, J., and Zwiehoff, G.  
Built-in logic block observation techniques.  
*In Test Conference Proceedings*, pages 37-41. October, 1979.
- [Puri 77] Puri, P.  
The functional tester: an aid to the designer.  
*In Test Conference Proceedings*, pages 64-80. 1977.
- [Seitz 80] Seitz, C.  
System Timing.  
*In Introduction to VLSI Systems*, chapter 7, pages 218-254. Addison-Wesley, 1980.

THE ROLE OF TEST CHIPS IN COORDINATING LOGIC  
AND CIRCUIT DESIGN AND LAYOUT AIDS FOR VLSI

Martin G. Buehler and Loren W. Linholm  
National Bureau of Standards  
Washington, DC 20234

ABSTRACT

This paper emphasizes the need for multipurpose test chips and comprehensive procedures for use in supplying accurate input data to both logic and circuit simulators and chip layout aids. It is shown that the location of test structures within test chips is critical in obtaining representative data, because geometrical distortions introduced during the photomasking process can lead to significant intrachip parameter variations. In order to transfer test chip designs quickly, accurately, and economically, a commonly accepted portable chip layout notation and commonly accepted parametric tester language are needed. In order to measure test chips more accurately and more rapidly, parametric testers with improved architecture need to be developed in conjunction with innovative test structures with on-chip signal conditioning.

1. INTRODUCTION

The increasing complexity of VLSI circuits is forcing the development of coordinated aids for circuit design and layout, in particular aids that can be used to predict the performance of circuits prior to their fabrication. Key elements in the development of design and layout aids are microelectronic test chips, parametric testers, and data analysis procedures. Since test chips provide the input data for the logic and circuit simulators and the chip layout aids, it is essential that test chips be developed in concert with the simulators and layout aids.

Microelectronic test chips\* have been used by the integrated circuit industry for many years. Typically, the test chips are substituted at several selected sites for integrated circuits on production wafers. The points in an integrated circuit production sequence where test chips can provide valuable information are illustrated in figure 1. This sequence shows the logic design sequence on the left and the circuit layout and fabrication

---

Contribution of the National Bureau of Standards, not subject to copyright.

\*In the authors' view, the term test chip is preferred over the term test pattern which can be confused with a sequence of electrical pulses used to test a circuit. The preferred term used to describe this latter effect is test vector.

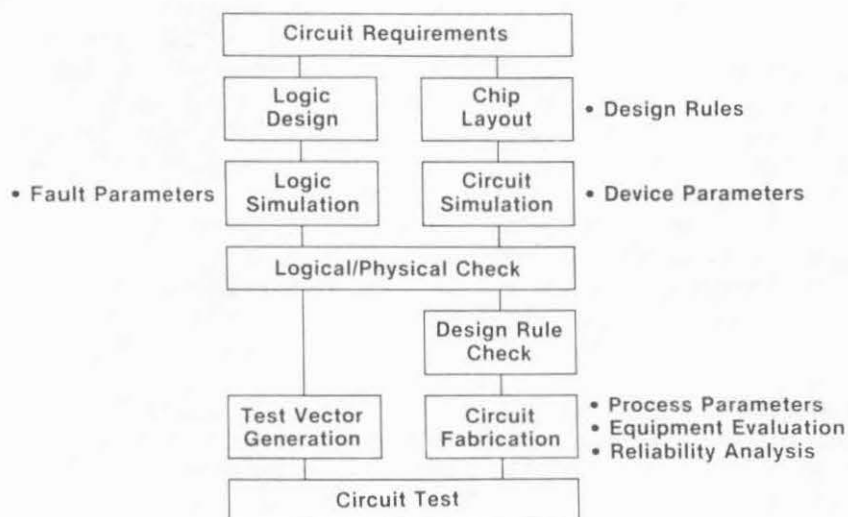


Figure 1. A simplified integrated circuit production sequence illustrating the points where test chips provide needed information.

sequence on the right. Before fabrication, cross checks are made to ensure that the physical design correctly implements the logical specification [1].

Traditionally, test chips have been used to supply process and device parameters and subcircuit data. In recent years, test chips have been used to evaluate such processing equipment as photomask aligners [2] and ion implanters. Also, reliability information has been obtained from test chips containing MOS capacitors which are analyzed for mobile oxide charge contamination and interface state densities [3]. In this paper we wish to highlight the role of the test chip as identified in figure 1 in circuit simulation, logic simulation, and chip layout.

## 2. CIRCUIT SIMULATION

The simulation of the dc and timing characteristics of a circuit is essential in identifying circuit design flaws prior to the fabrication of complex VLSI circuits. In this section, the circuit simulator requirements are discussed in terms of test structures for (a) acquiring device parametric data, (b) verifying the dynamic circuit performance capability of the fabrication process, (c) measuring intrachip parameter variations, and (d) evaluating the onset of small geometry effects.

Simulators such as SPICE [4] or MSINC [5] require device models, device parametric data usually collected from test chips, and geometrical layout information for the circuit being simulated. The device engineer "adjusts" various parameters in the simulator code so that the device models faithfully replicate the observed device performance. From discussions with industrial scientists, we found that the dc response and timing simulation of MOS digital logic circuits is judged as adequate for processes where both the design rules and fabrication procedures are stable. In such a stable environment, the devices can be modeled with the use of a mixture of empirical intuition and physical insight.

The predictive capability of circuit simulators is greatly reduced when the design rules change (e.g., devices become smaller) or when the process is changed significantly. For these circumstances more sophisticated device phenomena (e.g., short-channel effects) must be taken into account. In the long term, industry will need circuit simulation codes and data collection methodologies (based on test chips) that are easy to use and upgrade and can predict circuit performance as design rules and processes change.

A commonly accepted test circuit must be developed which can verify that the fabrication process is capable of producing circuits with correct dynamic (or timing) properties. Traditionally, ring counters which are characterized by their frequency of oscillation have been used for this purpose. However, the oscillation frequency can often not be adequately simulated (especially in MOS circuits) because many interdependent factors contribute to its magnitude. A candidate circuit should be easy to measure in chip form and its performance should be simple to evaluate.

As feature sizes become smaller, the ability to fabricate circuits with uniform features will become more difficult. As a result, the percentage

variation in device parameters will increase. This variation will make circuit simulation more difficult. It also poses a problem in the placement of test devices on a test chip, for a device located in one part of the chip can have significantly different characteristics from a supposedly identical device in another part of the chip. Such effects were observed by Ham [6] for the threshold of MOSFETs fabricated in silicon on sapphire. The variations can be either intrachip or interchip in nature.

To illustrate the importance of such parameter variations, we have measured the linewidth variations with the use of the cross-bridge test structure [7] shown in figure 2.\* Using the test structure shown in figure 2, a single mask test chip was prepared on a 10X master reticle. The test chip was composed of an 12 by 20 array of identical test structures with a design linewidth of 6  $\mu\text{m}$ . The final photomask was prepared from the master reticle by a step-and-repeat process. The photomask was used in conjunction with a contact printer and a photolithographic process to etch the test chip pattern into an 800-nm thick aluminum layer. The aluminum had been electron-gun evaporated and deposited on an oxide film thermally grown on a 2-in. (50.8-mm) diameter silicon wafer.

The linewidth variations shown in figure 3 are from a single row of test structures measured across the diameter of the wafer. The plot shown in figure 3 indicates that the linewidth varies periodically with the chip dimension of 250 mil (6.35 mm). This periodic or *intrachip* variation is superimposed on a nonperiodic or *interchip* linewidth variation due to those factors which affect the contact between the photomask and the photoresist-covered wafer. The periodic or intrachip linewidth variation is due to aberrations in the optics of the image repeater used to step and repeat the 10X reticle. Similar results have been reported for a 15- $\mu\text{m}$  line [8].

The absolute magnitude of the variations shown in figure 3 is independent of the magnitude of the linewidth. Thus, the impact of such linewidth variations on device characteristics is quite dramatic especially for small devices. The linewidth variation for the lines shown in figure 3 is about 13 percent. For 1- $\mu\text{m}$  lines, the variation would be 70 percent. These results illustrate the importance of the location within the test chip where "representative" device parameters are measured. Ultimately, the accuracy of the data entered into circuit simulators will be limited by both intrachip and interchip parameter variations.

The final item mentioned in this section concerns the development of test structures and evaluation techniques which tell when the models used in the circuit simulators are no longer valid due to the onset of small geometry effects. Of critical importance for future VLSI components are the

\*The linewidth is determined after measuring the sheet resistance  $R_s$  which is determined from  $R_s = (\pi/\ln 2)(\Delta V/I)$ , where  $I$  is the current forced between  $I_1$  and  $I_2$  and  $\Delta V$  is the voltage difference between  $V_1$  and  $V_2$ . The linewidth  $W$  is given by  $W = R_s L I^* / \Delta V^*$ , where  $L$  is the distance between the voltage taps shown in the figure,  $I^*$  is the current forced between  $I_1^*$  and  $I_2^*$ , and  $\Delta V^*$  is the voltage difference measured between  $V_1^*$  and  $V_2^*$ . A more detailed account of the measurement is given in reference [7].



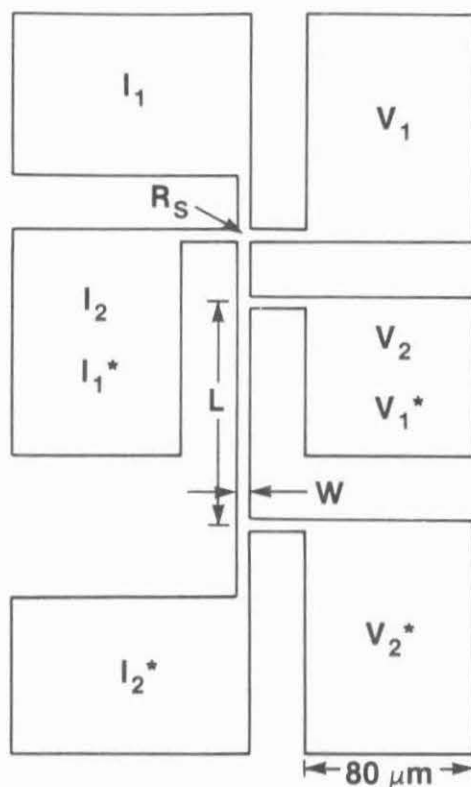


Figure 2. The cross-bridge test structure used to measure the sheet resistance and linewidth of a conducting layer.

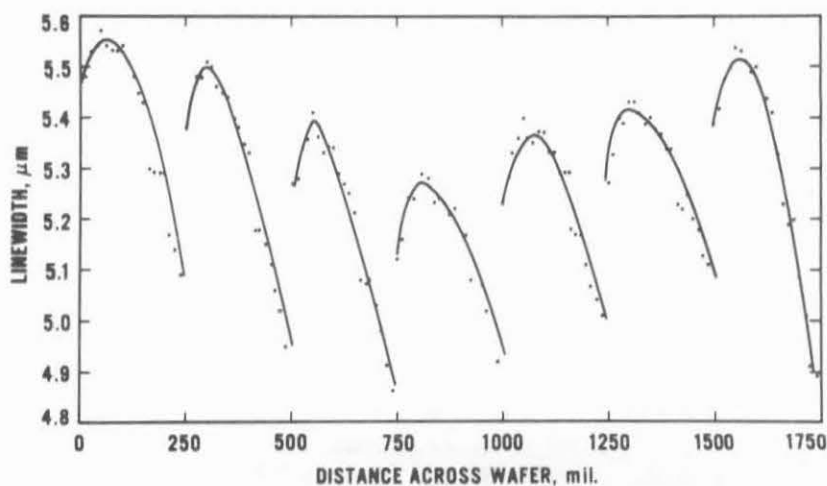


Figure 3. The variation of the linewidth as determined from an array of identical cross-bridge test structures as shown in figure 2. The period of the linewidth variation corresponds to the width of the test chip, 250 mil (6.35 mm). This periodic linewidth is due to aberrations in the optics of the image repeater used to step and repeat the 10X reticle.

evaluation of MOSFET short-channel effects and capacitor fringing field effects.

### 3. LOGIC SIMULATION

The logic simulation of the gate level performance of a circuit is essential in identifying logic design flaws prior to the fabrication of complex VLSI circuits. As shown in figure 1, the logic simulator can also be used to develop the test vectors used to test the circuit after fabrication. An accurate simulation depends on having proper fault models, correctly identified faults and their density, and detailed layout information.

An example of a test structure which can provide detailed information for fault simulation is shown in figure 4. This figure shows a MOSFET array which is composed of 100 devices where the gate is connected to the drain. This structure appears on test chip NBS-16 [9] which includes two *p*-channel and two *n*-channel MOSFET arrays. On a 3-in. (76.2-mm) diameter wafer, 380 arrays containing 38,000 MOSFETs were tested. The results shown in table 1 are from 26,760 MOSFETs located in the interior portion of the wafer. Both the fault location and the relative density of different fault types, both clustered and nonclustered, can be determined from the electrical data. A fault is considered to be clustered when two or more adjacent MOSFETs containing the same fault type are detected in an array. As seen in the table, the most frequent fault was #8, a combination of excessive leakage current and low breakdown voltage.

A major limitation of present logic simulators is their inability to properly model faults other than classical faults. The latter comprise those faults where signal lines are either shorted to the power supply (stuck-at-one), shorted to ground (stuck-at-zero), or simply open circuited. Sievers [10] has recently shown how the classical faults and open and short faults can be modeled for both *n*MOS and CMOS circuits. Any of the first five faults listed in table 1 could lead to a classical, open, or short fault. But the total of these five faults is only a small fraction of the total of the nonclassical faults, #6 through #13. Future logic simulators must have the ability to model such nonclassical faults in order to enable more realistic circuit fault simulations.

When using a logic simulator, it is essential that there be a one-to-one correspondence between the logic representation and the physical layout [11]. For example, the logic diagram shown in the upper part of figure 5 bears little resemblance to the accompanying circuit schematic. To illustrate, the wire D joining the two gates is not uniquely found in the circuit schematic. To perform a fault simulation where a classical fault (stuck-at-one or stuck-at-zero) is introduced on this wire is a meaningless exercise.

### 4. CHIP LAYOUT

Chip layout is a very important part of a VLSI design system because the layout influences all other parts of the system. For example, the layout data set can be used to derive the geometrical data needed by the circuit and logic simulators. In addition, the choice of layout notation can restrict

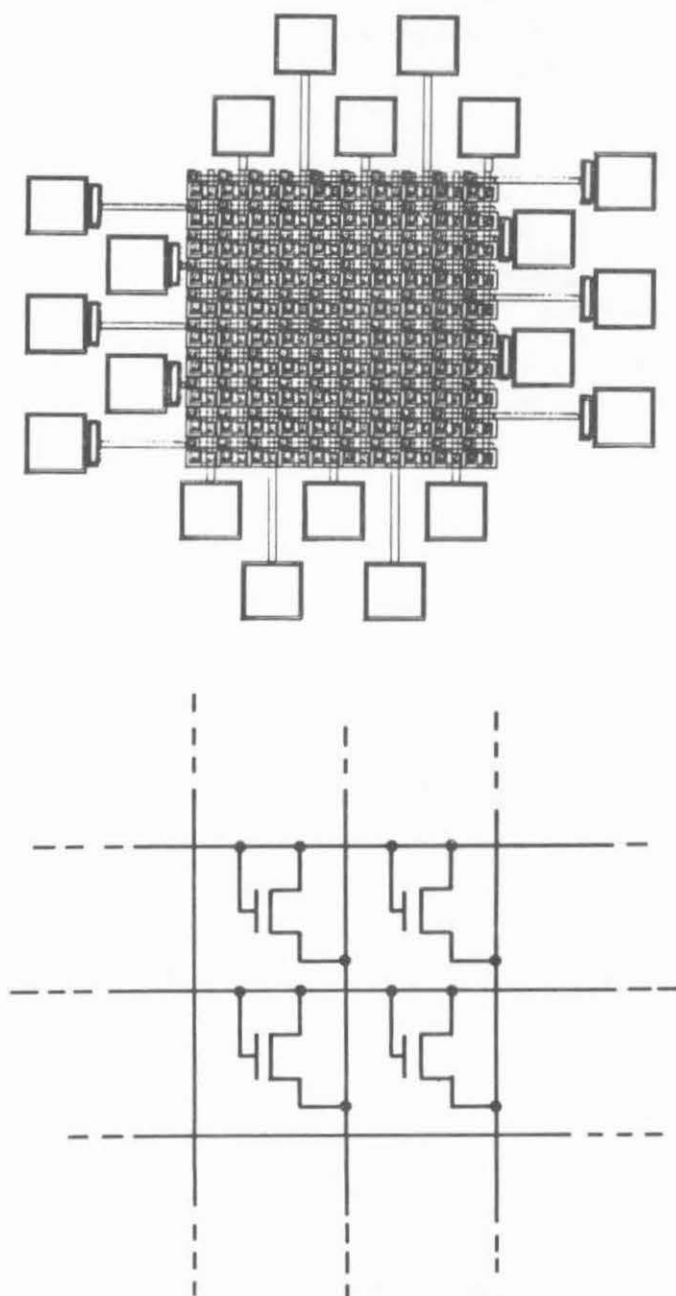


Figure 4. MOSFET array test structure consisting of 100 individually addressable transistors shown above. A schematic diagram is shown below.

Table 1. MOSFET Array Test Results.

Number	Fault	Number of Nonclustered Faults	Number of Clustered Faults
1	Poly void/break	4	0
2	Epi void	0	1
3	Metal void/break	0	0
4	Metal bridge	1	0
5	Gate short	4	0
	$V_T$ $I_L$ $V_B$		
6	- - L	5	0
7	- H -	25	0
8	- H L	62	0
9	L - -	1	1
10	L H -	0	1
11	L H L	0	1
12	H - -	2	2
13	H H L	0	1

$V_T$  = threshold voltage;  $I_L$  = leakage current;  $V_B$  = breakdown voltage.  
H = parameter too high; L = parameter too low.

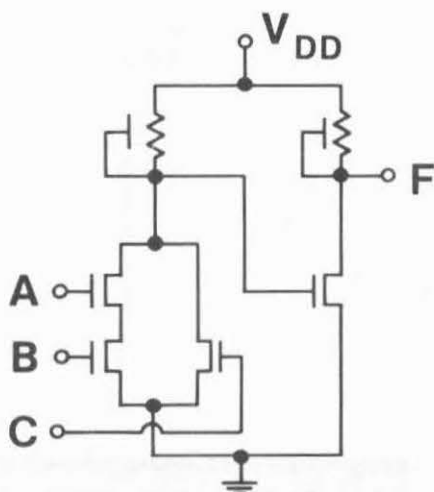
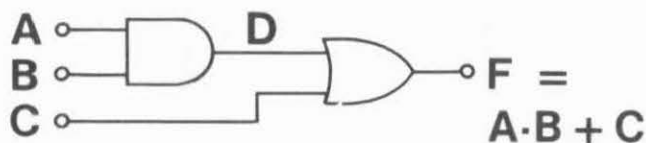


Figure 5. Lack of correspondence between the logic diagram and the circuit schematic in that wire D does not appear uniquely in the circuit diagram.

the portability of the design. In the authors' view, the chip layout methodology must allow for design portability so that circuits can be fabricated in facilities with different design rules. The idea is to supply different fabrication facilities with a chip layout description that can be adjusted to meet the design rules for that particular facility. This idea requires that the manufacturer adjust the chip layout pattern as it appears on the photomask to allow for changes in feature sizes during fabrication so that features appear with specific dimensions on the finished circuit. One layout representation that appears to be a candidate for such portability is symbolic notation [12].

An additional need in chip layout is to develop test structures that can be used to establish layout design rules. Structures useful for this purpose are known as random fault test structures where a feature (e.g., metal-to-silicon contact) is repeated many times within an array [13]. Arrays with different numbers of features are fabricated and tested for opens or shorts. The number of good features per fault characterizes the process. The test structures to be developed must be designed so that the intended fault is measured. Good design practice dictates that each probe pad used to contact an array accommodate two probes (Kelvin contacts) so that one can be assured that the probes are making contact [8]. In addition, the arrays must be designed so that interferences between other parts of the array are eliminated [14].

## 5. TEST CHIP

Despite the long history of test chip usage by the integrated circuit industry, there has been relatively little emphasis placed on the development of such chips. To make better use of test chips in the future, one must develop a coordinated metrological system including advanced parametric testers, commonly accepted parametric tester language, improved microelectronic test structures, and efficient information-handling procedures.

The first multifunction parametric tester specifically designed to measure test chips was commercially available in 1978 and now there are at least three systems which can be purchased [15]. The availability of these systems greatly enhances the use of test chips in production wafer-probe environments. The commercially available systems typically have a system architecture that is based on the mechanical switch matrix as seen in figure 6. The accuracy and precision of such systems is limited by noise introduced by the switch matrix and long leads. In addition, measurement times can be long when measuring low-level quantities. The authors feel that these limitations can be overcome by changing the system architecture to the pin-electronics approach [16] as seen in figure 6. Here the stimulus/measure (S/M) devices are physically located close to the wafer probes. In addition, the number of wafer probes can be profitably increased (20 to 40 in the example) because test structures addressed by a wafer-probe array can be measured simultaneously. Overall test times for the pin-electronics approach should be significantly reduced [17].

A commonly accepted parametric tester language is needed to facilitate the rapid, accurate, and economical transfer of test chip measurements. As is

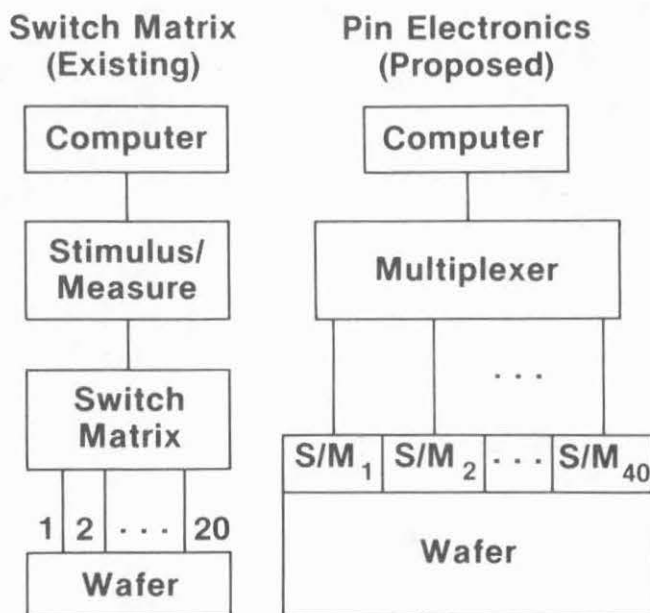


Figure 6. System architecture of multifunction parametric test equipment (existing and proposed [17]).

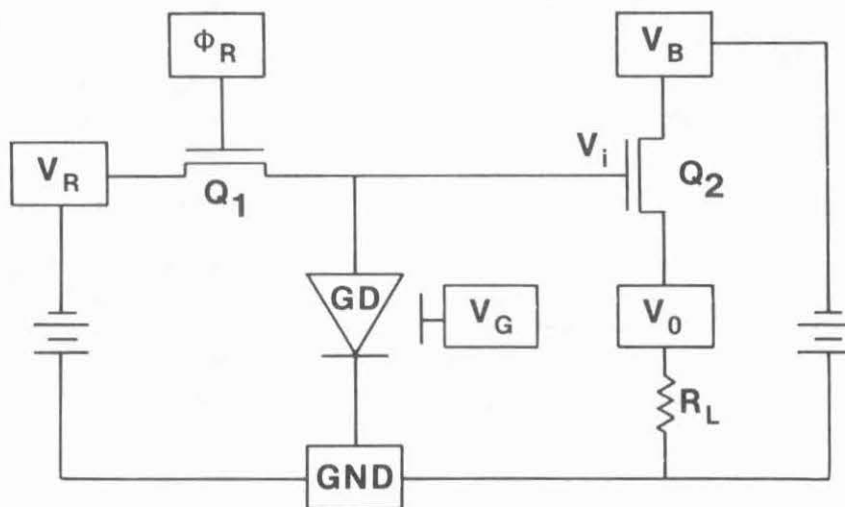


Figure 7. Schematic diagram of an integrated gated-diode electrometer. The boxes represent probe pads. The off-chip components are the two dc supplies and the resistor.

well known, software development costs are very expensive. There is an IEEE standard test language named ATLAS (Abbreviated Test Language for All Systems) [18] that was originally developed by the commercial airlines industry for testing avionics packages. Recently, a software package was developed which translates ATLAS into a test language used in the testing of digital printed circuit boards [19]. Currently, there is no comparable standard test language for parametric chip testers.

When developing new test structures, one must decide if there is a measurement advantage to be gained by incorporating a portion of the tester into the test structure. Such an advantage has been found with the integrated gated-diode electrometer shown schematically in figure 7. Low-level diode leakage currents can be determined by measuring the time decay of the output voltage  $V_O$  resulting from the momentary application of reverse bias voltage  $V_R$  to the gated diode GD through the MOSFET switch  $Q_1$ . The internal gated-diode current  $I$  is determined from the expression [20]:

$$I = (C/\beta) (-dV_O/dt) ,$$

where  $C$  is the diode capacitance and  $\beta$  is the incremental gain of MOSFET  $Q_2$ . The diode capacitance can be determined from  $C = \epsilon A/W$  where  $\epsilon$  is the dielectric constant for silicon,  $A$  is the area of the diode, and  $W$  is the width of the depletion region. For a one-sided step junction,  $W = [2\epsilon(V_i + V_b)/(qN)]^{1/2}$  where  $V_i$  is the diode voltage,  $V_b$  is the built-in voltage,  $q$  is the electronic charge, and  $N$  is the dopant density. The dc gain of MOSFET  $Q_2$  is determined from  $\beta = \Delta V_O/\Delta V_i$  which is evaluated by closing the MOSFET switch  $Q_1$  ( $V_i = V_R$ ) and measuring  $V_O$  at two different values of  $V_R$ . The expression above assumes that the capacitance of the gated diode  $C$  is large compared to the gate-source capacitance of MOSFET  $Q_2$ . The test structure design shown in figure 8 obeys this restriction. The off-chip output resistor  $R_L$  shown in figure 7 is replaced by an on-chip load MOSFET  $Q_3$  with its gate connected to the  $V_O$  point so that it operates in the saturated mode.

Another integrated test structure has recently been reported by Iwai and Kohyama [21]. This structure is shown schematically in figure 9. Here, the unknown capacitance

$$C_X = C_R [\beta(v_i/v_{oa}) - 1] ,$$

where  $C_R$  is a known reference capacitor,  $\beta$  is the ac gain of the output MOSFET,  $v_i$  is the rms value of the ac input signal, and  $v_{oa}$  is the output at  $v_o$  for  $v_i$  connected to point "a." To measure  $C_X$ ,  $v_i$  is applied to point "a" and the MOSFET switch  $Q_1$  is opened by properly biasing  $\phi$ . The ac gain  $\beta$  is determined from  $\beta = v_{ob}/v_i$  where  $v_{ob}$  is the output at  $v_o$  for  $v_i$  connected to point "b." In this measurement the MOSFET switch  $Q_1$  is closed by properly biasing  $\phi$ . This structure is useful in determining the value of the small capacitances typical of VLSI device geometries by connecting many capacitors into an array.

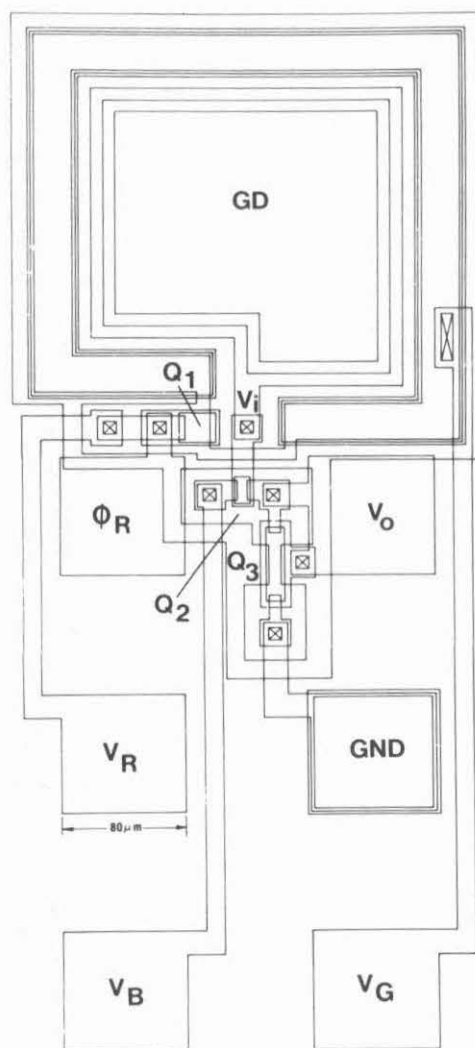


Figure 8. The integrated gated-diode electrometer test structure shown for a polysilicon-gate technology.



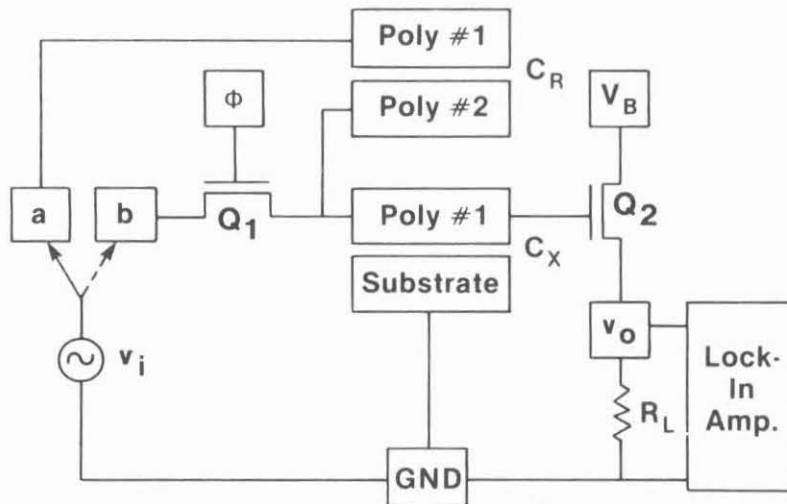


Figure 9. Schematic diagram of an integrated precision capacitance meter. The off-chip components are  $V_i$ ,  $R_L$ , and the lock-in amplifier.

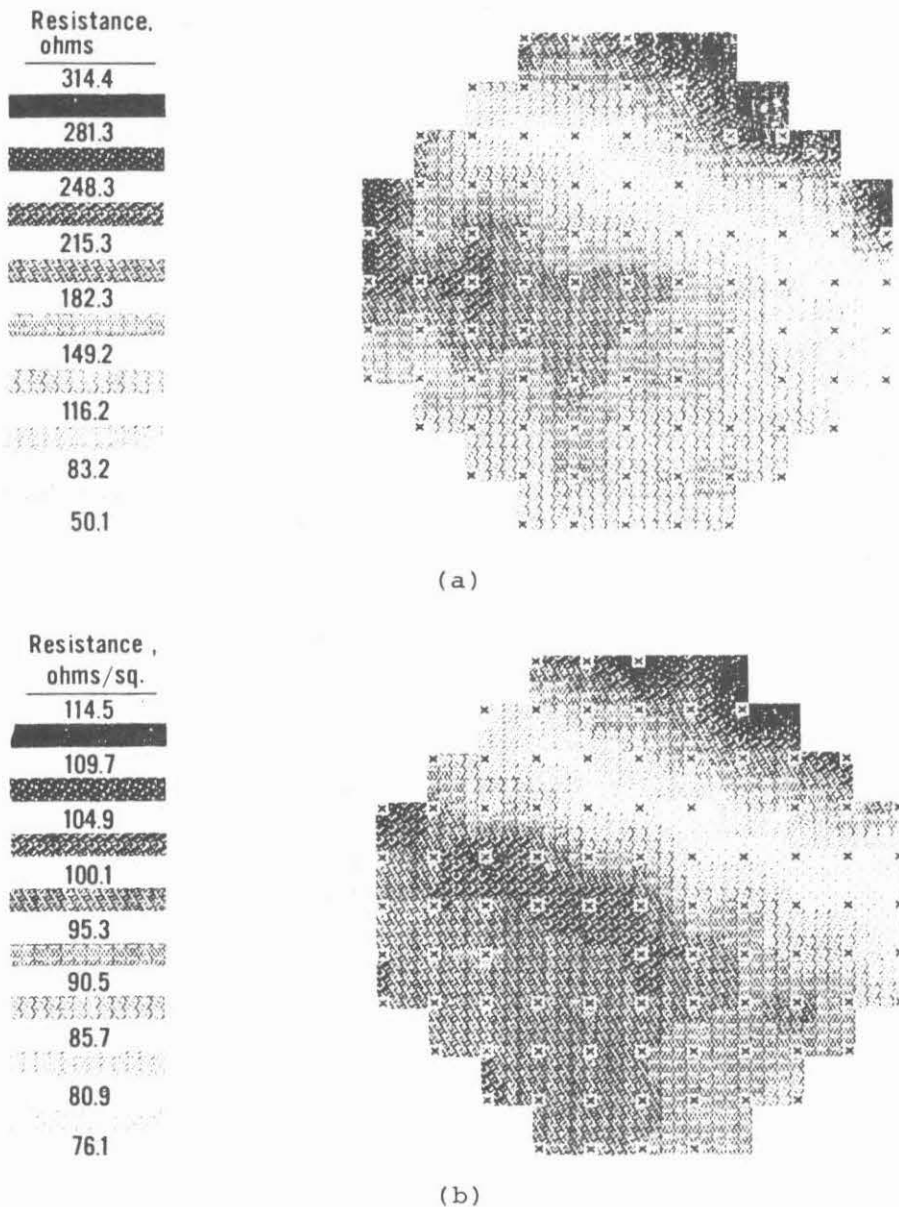


Figure 10. Wafer maps of the metal-to- $n^+$  contact resistance (a) and the  $n^+$  sheet resistance (b). The high correlation between these two parameters leads to the conclusion that excessively high contact resistance was due to the lack of adequate control of a phosphorus implant process step.

Once the information has been collected from test chips, the data must be quickly analyzed to be of benefit. Wafer maps of a spectrum of parameters are crucial in identifying problems [22]. For example, a contact resistance problem was identified and correlated to variations in sheet resistance as shown in figure 10 [9]. The correlation was identified by observing a high correlation of these parameters from a host of other parameters [23]. The manufacturer of this wafer would not have identified this problem because his procedures called for only two test chips on each product wafer. Other information-handling techniques have been developed over the years [24]. The industry needs to fully utilize existing techniques and to search for more useful and efficient techniques.

## 6. CONCLUSION

Test chips have a role that goes beyond their traditional role involving process or subcircuit evaluation. The additional role involves supplying the data for the logic and circuit simulators and in setting the design rules for chip layout methods. The development of design aids and test chips must be coordinated to provide a well-integrated design system. In addition, new parametric testers are needed that can quickly supply data of increased accuracy. The development of such testers must be coordinated with test structure development so as to take advantage of on-chip signal processing. In addition, effective data-handling techniques (e.g., wafer mapping and data management) need further development so as to rapidly reduce data to a useful form. In order to transfer test chip designs quickly, accurately, and economically, a commonly accepted portable chip layout technique and a commonly accepted parametric tester language are needed.

## 7. ACKNOWLEDGMENTS

The authors are indebted to Charles Wilson, Tom Leedy, and others for their insights into the needs for parametric data for circuit simulators. The authors appreciate Gary Carver's contribution of the integrated gated-diode electrometer design as shown in figure 8 and the critical reviews of Barry Bell, Murray Bullis, and Christoph Witzgall.

## REFERENCES

1. E. I. Muehldorf, Test Pattern Generation as a Part of the Total Design Process, *1978 Semiconductor Test Conference*, pp. 4-7 (1978).
2. T. F. Hasan, D. S. Perloff, and C. L. Mallory, Test Vehicles for the Measurement and Analysis of VLSI Lithographic and Etching Parameters, *Semiconductor Silicon 1981* (to be published).
3. K. H. Zaininger and F. P. Heiman, The C-V Technique as an Analytical Tool, *Solid State Technology* 13, 49-56 (May 1970).
4. L. W. Nagel, SPICE2: A Computer Program to Simulate Semiconductor Circuits, Memorandum No. ERL-M510, Electronics Research Laboratory, University of California, Berkeley, California (May 9, 1975).
5. T. K. Young and R. W. Dutton, Mini MSINC - A Minicomputer Simulator for MOS Circuits with Modular Built-In Model, *J. Solid-State Circuits* SC-11, 730-732 (1976).
6. W. E. Ham, Intrachip and Spatial Parametric Integrity: An Important Part of IC Process Characterization, *1977 IEDM*, 406-409 (1977).
7. M. G. Buehler, S. D. Grant, and W. R. Thurber, Bridge and van der Pauw Sheet Resistors for Characterizing the Linewidth of Conducting Layers, *J. Electrochem. Soc.* 125, 650-654 (1978).
8. M. G. Buehler, The Use of Electrical Test Structure Arrays for Integrated Circuit Process Evaluation, *J. Electrochem. Soc.* 127, 2284-2290 (1980).
9. L. W. Linholm, *Semiconductor Measurement Technology: The Design, Testing, and Analysis of a Comprehensive Test Pattern for Measuring CMOS/SOS Process Performance and Control*, NBS Spec. Publ. 400-66 (to be published).
10. M. W. Sievers, Approaching Fault Tolerant VLSI, *Government Microcircuit Applications Conference, Technical Digest*, 256-259 (1980).
11. H. S. DeMan, Computer-Aided Design for Integrated Circuits: Trying to Bridge the Gap, *J. Solid-State Circuits* SC-14, 613-621 (1979).
12. R. P. Larsen, Symbolic Layout System Speeds Mask Design for ICs, *Electronics* 51, 125-128 (July 20, 1978).
13. A. C. Ipri, Impact of Design Rule Reduction on Size, Yield, and Cost of Integrated Circuits, *Solid State Technology* 22, 85-89 (February, 1979).
14. M. A. Mitchell, private communication.

*The Role of Test Chips in Coordinating Logic and Circuit Design and Layout Aids for VLSI*

15. C. Chrones, Parametric Test Systems for Wafer Processing, *Semiconductor International* 3, 113-122 (October 1980).
16. P. C. Jackson, Optimization of ATE Switching/Multiplexing, *1980 IEEE AUTOTESTCON*, 242-246 (November 1980).
17. M. G. Buehler and D. S. Perloff, Microelectronic Test Chips and Associated Parametric Testers: Present and Future, *Semiconductor Silicon 1981* (to be published).
18. IEEE Guide to the Use of ATLAS (J. Wiley, New York, 1980).
19. T. Lapetina and D. Schneider, An ATLAS Implementation for a Commercial Tester, *1980 IEEE AUTOTESTCON*, 5-7 (November 1980).
20. G. P. Carver and M. G. Buehler, An Analytical Expression for the Evaluation of Leakage Currents in Integrated Gated-Diode Electrometers, *IEEE Trans. Electron Devices* ED-27, 2245-2252 (1980).
21. H. Iwai and S. Kohyama, Capacitance Measurement Technique in High Density MOS Structures, *1980 IEDM*, 235-238 (December 1980).
22. J. M. Charles and M. W. Lantz, Applications of High Speed Data Acquisition for Semiconductor Device Yield Analysis, *IEEE Trans. Electron Devices* ED-27, 2299-2303 (1980).
23. L. W. Linholm, R. L. Mattis, and R. C. Frisch, Characterizing and Analyzing Critical Integrated Circuit Process Parameters, *Semiconductor Silicon 1981* (to be published).
24. T. Cave and D. Smith, Data Reduction and Display in Automated Test Systems, *Computer Design* 17, 161-172 (May 1978).



*INNOVATIVE LSI DESIGNS SESSION*

*Chairperson: GERALD J. SUSSMAN*  
*ASSOCIATE Professor of Electrical Engineering*  
*and Computer Science*  
*Massachusetts Institute of Technology*





## Bit-Serial Inner Product Processors in VLSI

*Misha R. Buric*

Bell Laboratories  
Murray Hill, New Jersey 07974

*Carver A. Mead*

California Institute of Technology  
Pasadena, California 91125

### 1. Introduction

Many problems in signal and image processing, pattern recognition, and feedback systems involve models with vector variables. Besides vector addition and multiplication by a scalar, an inner product of vectors is a basic arithmetic operation in these models. It is computationally most demanding, so that there is considerable interest in finding ways to speed up its implementation. Array configurations of simple processors for performing vector and matrix operations have been extensively reported. A number of ideas can be found in [1], [2], [7], [8], and their references.

In this paper we describe a bit-serial pipelined implementation of an inner product processor, and related interconnections of a number of such processors on a single chip. We argue that bit-serial computational models are particularly suited for VLSI, because of relatively inexpensive communication links and arithmetic processing elements, in terms of the area occupied on silicon. Sixteen inner product processors, described here, may be easily placed on a single 40-pin chip in today's NMOS technology with a 2 micron lambda. Similar arguments for bit-serial arithmetic were used in [3], in a description of a design of a general purpose massively parallel processor.

Generally, all multiprocessor schemes can be divided into two classes with respect to the interconnection patterns among processing elements. Static communication links characterize those array configurations in which a fixed algorithm is executed repeatedly and synchronously with the input data flow. These structures are especially useful if the input information is being continuously provided by sampling some real world variables, and the purpose of the processing is to provide a compressed version of the data, or to transform it into another representation. Many examples can be found in speech and image processing. This structure and its variations is examined in this paper. Another, more general class of multiprocessor schemes involves flexible communication interconnections among the processors through switched networks, [7].

### 2. Basic Processors and their Interconnections

An inner product of two  $n$ -dimensional vectors,  $x$  and  $y$ , is defined by

$$z = x^T y = \sum_{i=1}^n x_i y_i \quad (1)$$

where

$$x^T = [x_1 \ x_2 \ \dots \ x_n] \text{ and } y^T = [y_1 \ y_2 \ \dots \ y_n]$$

We use a convention that all vectors are column vectors, and that  $T$  denotes a transpose operation. A transposed column vector is a row vector. It is assumed here that all the vector elements are integers. The equation (1) can be rewritten in an iterative form as follows:

$$z_i = x_i y_i + z_{i-1}, \quad i=1, 2, \dots, n \quad (2)$$

$$z_0 = 0, \quad z = z_n$$

There are a number of ways to implement this equation, ranging from a single multiplier-accumulator combination, to an array of  $n$  processors.

In the first case, which may be called an iteration in time, the  $2n$  operands are fetched two at the time, they are multiplied together, and added to the previously accumulated value. This requires  $n$  steps, assuming that the pipeline registers are used to enable overlapping of the operations.

On the other side, an iteration in space is characterized by a pipelined array configuration of  $n$  processors that operates on  $2n$  operands simultaneously, and provides a new result every cycle. The  $i$ -th processor is assigned to the  $i$ -th elements of the input vectors, and to the  $(i-1)$ -st partial sum, and it produces the  $i$ -th partial sum. Due to pipelining, a processor may receive new operands every cycle.

Iteration in space will be considered first, because it provides the necessary throughput for large vector sizes. In a VLSI implementation of this scheme, the cost of arithmetic elements, and the communication cost of providing  $2n$  operands and interconnecting individual processors, would be prohibitive if we were to use word-parallel arithmetic.

Bit-serial arithmetic and communication, however, offers a viable alternative for two reasons. First, the arithmetic components and the communication lines occupy much smaller area on silicon. Therefore, a larger number of basic processors may be integrated on a single chip. Second, the nature of inner product computation requires more bits of precision for larger vector sizes. A fixed-word parallel arithmetic is not suitable for a flexible precision control, since the overflow conditions may occur for larger vectors, unless a sufficiently large adders are provided in advance. On the other side, bit-serial addition does not suffer from this problem, since the precision may be maintained arbitrarily high with the same hardware.

In this text we use  $b^{-1}$  as a bit-delay operator, analogously to  $z^{-1}$  which we reserve for signal processing applications. This notation is convenient for treating bit-serial systems, because there may be various delays in a complex array of serial elements, so that a systematic treatment of path delays is important.

Our implementation of a single inner product processing element is shown in Figure 1.

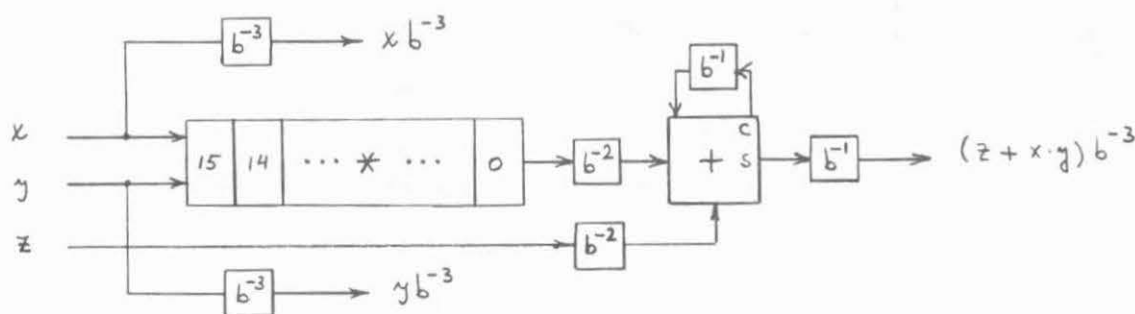


Fig. 1

It consists of a modular bit-serial multiplier, (two's complement, 16-bit), and a single-bit carry-save adder. The input variables  $x$ ,  $y$ , and  $z$  enter the processor with the least significant bit first, and the result starts appearing three bit times later. The computation is synchronized by a control bit, that is applied simultaneously with the LS bits of the operands. The processor provides 31 bits of the result. An additional 0 bit is inserted between the result and the LS bit of the next product. The input operands are padded with 16 zeros to the left of the most significant bit, so that new operands may be applied every 32 clock cycles.

Clocking is two-phase, such that phase one controls all data transfers, and all logic functions are evaluated in phase two. The delays  $b^{-1}$  imply shift-register pipeline stages.

There are some variations of the circuit in Figure 1 that are application dependent. The delayed input variables do not have to be provided at the output in some array configurations. Also, we will show later in the text that the adder may be connected to a pair of multipliers instead as shown in Figure 1.

The size of a single inner product processor was 2720 by 155 lambda with a linear layout, and 680 by 620 lambda in a rectangular configuration. There was no attempt made to minimize the size of the basic multiplier cell. Despite this, it is feasible to place sixteen such processors on a 40-pin chip, but the limitation in this and future implementations is not so much the area, as much as the number of external connections.

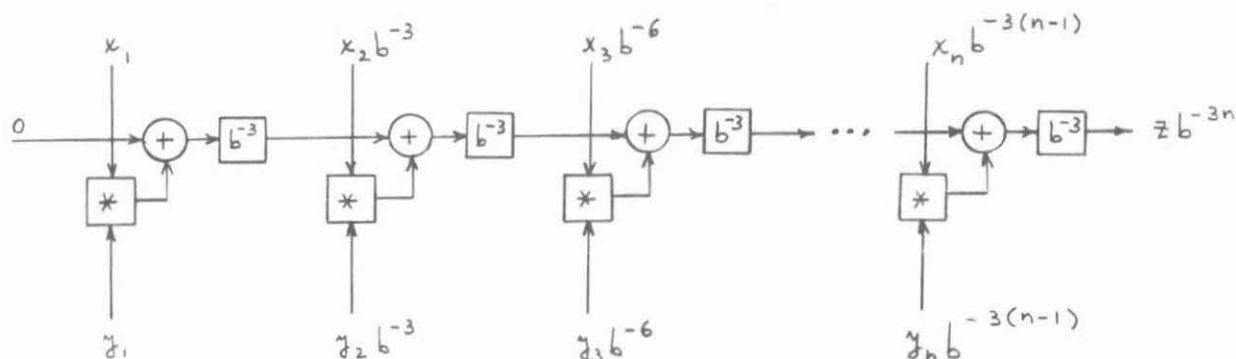


Fig. 2

This brings up the main question: what is the best interconnection among processors on a chip, which makes further combinations of such chips most flexible?

A linear array, as in Figure 2, has many desirable properties for a number of applications. The number of pins for a 16 processor combination is less than 40, and the array may grow arbitrarily large. The expression for an inner product of two large vectors has the same iterative form in terms of inner products of their smaller parts, as in equation (2). That is, if  $x$  and  $y$  are two vectors of size  $M$ , we can partition them into  $K$  groups of smaller vectors of size  $n$ , so that the inner product  $x^T y$  may be evaluated as follows:

$$x^T = [c_1^T \ c_2^T \ \cdots \ c_K^T], \quad y^T = [d_1^T \ d_2^T \ \cdots \ d_K^T]$$

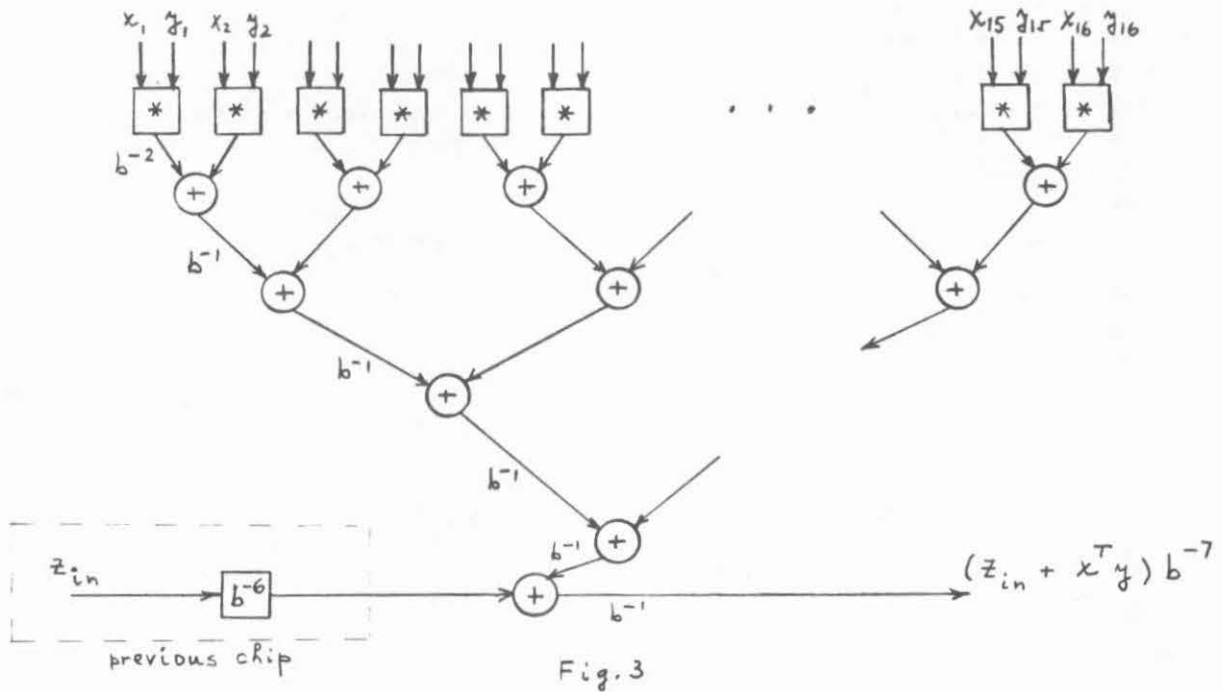
$$z = x^T y = c_1^T d_1 + c_2^T d_2 + \cdots + c_K^T d_K$$

$$z_i = c_i^T d_i + z_{i-1}, \quad i = 1, 2, \dots, K,$$

$$z_0 = 0, \quad z = z_K$$

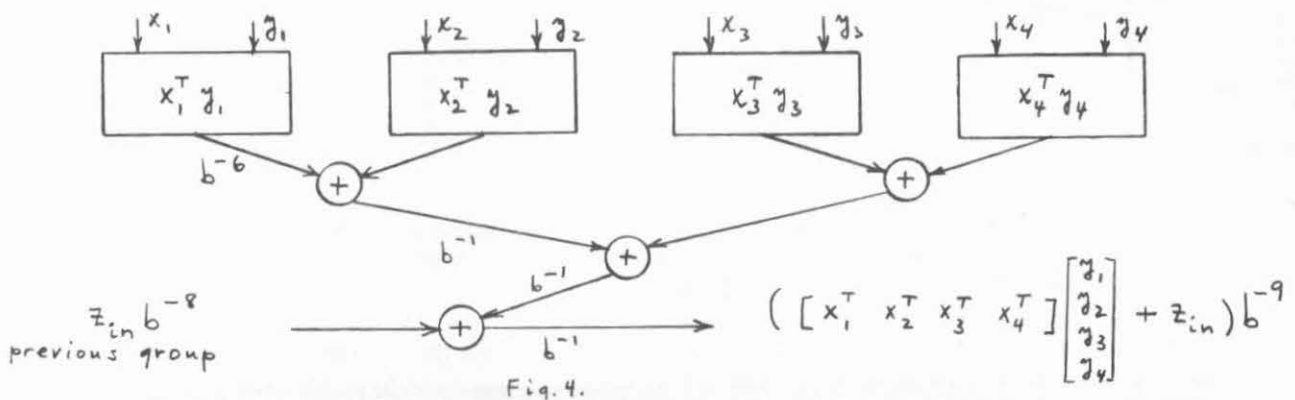
If each partial inner product of size  $n$  is computed in a separate chip, then the complete result is obtained by a linear connection of the  $K$  chips. The throughput rate remains the same, one inner product per 32 clock cycles regardless of the vector sizes. Of course, extra cycles may be inserted between two products if there is a need for the overflow control in the adder stages. In Figure 2 the variables  $x_i, y_i$  are integer elements of vectors  $x$  and  $y$ , which enter the processor bit serially. The expression  $x_i b^{-j}$  means that the integer  $x_i$  is delayed by  $j$  bit cycles. Notice that each element of the vectors is delayed by  $b^{-3}$ , with respect to the previous element. This skewing does not pose any conceptual difficulties, but for practical reasons it would be easier to apply all elements simultaneously, without any bit delays among the operands. A possible solution is to include an appropriate shift register delay at the input of each section, but this would increase the area of the chip.

A better form of a linear array of basic inner product processors, suggested in [8] for word-parallel arithmetic, is shown in Figure 3. This configuration does not require any delays



among the vector elements, and produces the initial product with the delay of only  $b^{-6}$  in a sixteen processor chip. Here, the adders are organized in a tree structure, and an additional carry-save adder is added to the chip, with no internal connections. The purpose of this adder is to allow interconnections of an arbitrary number of chips for larger inner products. The pipeline registers are assumed to be included in each adder, and the corresponding delays are shown externally.

A connection of individual 16 element chips for operating on larger vectors is shown in Figure 4. Summation of the partial inner products is again done by a tree structure of the adders, which are obtained from a pool of free adders.



In Figure 4 each  $x_i$  and  $y_i$  are 16 dimensional vectors.

This will be the main scheme in further discussion. It is sufficient for applications such as FIR and IIR filtering, matrix multiplications, vector convolutions and others.

Alternatively, the inner product processors may be connected together in a hexagonal array suggested in [1] and [2] for matrix operations. In this case, every processor has three

inputs and three outputs, but a chip with sixteen processors, shown in Figure 5, has two pairs of four inputs for the vector variables, two pairs of four delayed outputs of the same variables, and seven inputs and outputs for the result variables. In this case the input variables would be provided at the outputs delayed by  $b^{-3}$ , as in Figure 1. Larger hexagonal arrays may easily be constructed out of these sixteen processor chips, if each is viewed as a new basic element in a hierarchy. Even though this approach addresses a very important issue of the data flow through the network simultaneously with the computation, it requires more complex synchronization of the input operands.

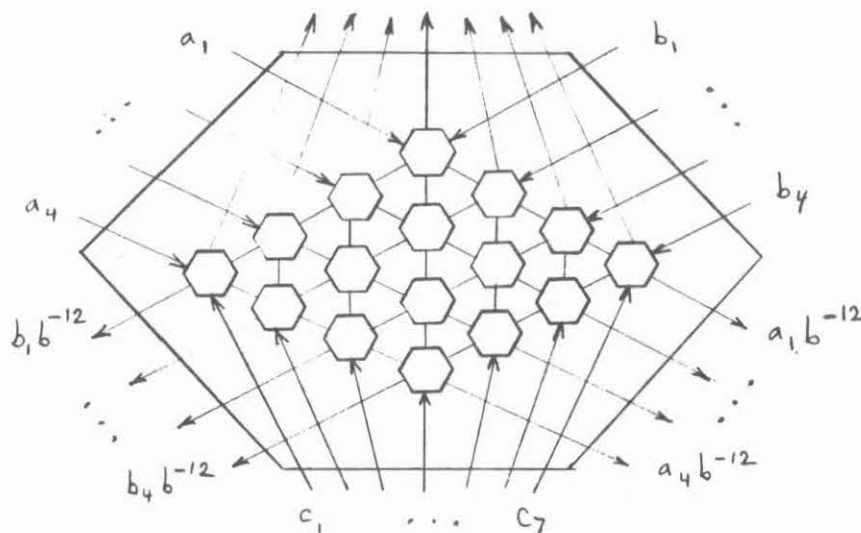


Fig. 5

### 3. Data Flow Control

It was assumed so far that an inner product of two vectors was computed by iterating  $n$  processors in space. The  $2n$  operands were supplied simultaneously bit-serially, and due to pipelining there was no delay between consecutive results of a series of inner products. Therefore, the computational throughput was adjusted to match the input data rate at the expense of more processors. This approach can be extended to matrix products, because they consist of a number of inner products. First, multiplication of an  $n$ -dimensional vector by a matrix, ( $m$  by  $n$ ), can be accomplished by  $m$  arrays of inner product processors, each consisting of  $n$  basic elements. One operand to all arrays is the vector, and the second operand is one row of the matrix for each array. Each array computes one component of the result simultaneously with others. Hence the throughput still remains the same as in the case of a single inner product. Similar structure can be used for a product of two matrices, ( $m$  by  $n$ ) and ( $n$  by  $k$ ), where  $mk$  linear arrays of dimension  $n$  compute  $mk$  results simultaneously.

In applications of inner product arrays outlined above there is a need for a data flow network that connects the source of operands with the computational structure, and provides for internal data flow during the operation. This is also important for purpose of matching the input data rate with the processing throughput. For example, in an FIR filter application a new data sample may be provided every  $T$  microseconds, and a filter has to perform one inner product of length  $N$  on two vectors. One vector consists of  $N$  previous samples, and the other is composed of filter coefficients. The result of the inner product is output once per period  $T$ . Let  $x_k$  be the input sample, and  $z_k$  the output value at time  $k$ . Then:

$$a^T = [a_0 \cdots a_{N-1}] \quad , \quad x^T = [x_k \ x_{k-1} \ x_{k-2} \ \cdots \ x_{k-N+1}]$$

$$z_k = a^T x = \sum_{i=0}^{N-1} a_i x_{k-i}$$

Therefore, both input and output are single integers, but the processor array operates on two vectors. In addition, the  $x$  vector has to be updated every sample time, in such a way that all the components change their position by one place:

$$x_{k+i+1} = x_{k+i}, \quad i = 0, 1, \dots, N-2$$

with the new sample becoming  $x_k$ . This is a shifting operation, suggesting a set of shift registers for this application. Now, if our inner product processor generates a result in  $\Delta t$  microseconds,  $\Delta t$  being much smaller than sampling period  $T$ , we can use a smaller processor, with only  $N \frac{\Delta t}{T}$  basic processors, but there has to be a mechanism for accumulating partial results within one inner product and cycling through all partial vectors of this smaller size.

This simple example is an illustration of a problem which contains a combination of communicational and computational complexity. A standard measure of computational complexity in this case would indicate that the problem is solvable in  $O(N)$  time, and since we can use an  $N$  processor array it becomes an  $O(1)$  problem. However, there are additional communication costs of providing  $2N$  operands, and performing  $N$  data exchanges. Also, in a VLSI implementation of this example it would be only sensible to provide all data movements within the same chip that contains the inner product processor, so that there is a single external connection for input and output.

This is typical for many algorithms with vector variables. Each operand interacts with a number of other operands before the computation is completed. Another example is a convolution of two vectors of size  $N$ , which requires  $2N$  inner products of the same vectors, but one of them is shifted each time. If there is a reasonable restriction that the data be brought into a VLSI vector processor only once, which minimizes the number of interactions with the external world, then there has to be some data storage on the chip with a flexible data exchange scheme. This issue prompted a hexagonal array approach in [1] and [2], in which the data storage and flow takes place in each basic processing element, and the topology of a network is tailored for the problem.

Here, we examine another alternative that seems to be well suited for bit-serial vector processing. Consider a shift-register element that has two inputs, horizontal and vertical, and one output, Figure 6. It can shift the data either horizontally or vertically, as determined by a shift control signal. Next consider a standard shift-register of length  $N$ , say 16, which consists of  $N-1$  standard cells and one two-input cell, shown in the same figure.

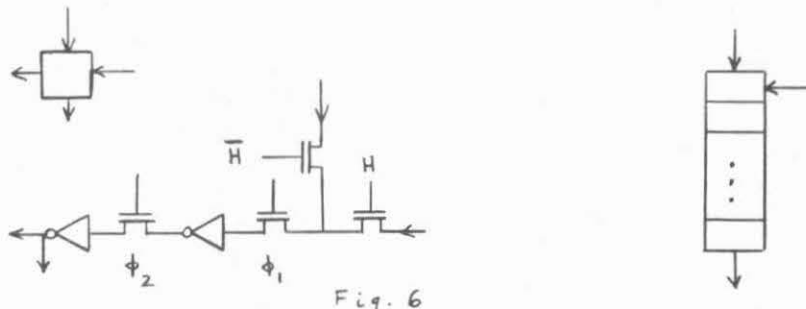


Fig. 6

A set of such shift registers can now be connected in a storage array with a two-dimensional shift capability. To demonstrate an application of such an array with an inner product processor let us consider an FIR filter implementation.

Suppose the filter is specified at 512 points, and the sampling period is 100 microseconds. A conservative estimate of the inner product performance is 3 microseconds per product. Therefore, a 16 processor array is sufficient for computing inner products of 512-dimensional vectors by iterating in time 32 times. A diagram of this configuration is shown in Figure 7, for



a smaller array. In order to iterate in time, a bit-serial accumulator is provided. There are two sets of register arrays, data and filter registers. The data registers are connected vertically in 16 circular groups, and horizontally into a linear array. In a case of a constant filter the filter registers may be connected in the same way as the data registers, even though there are applications in adaptive filtering where a different connection would be used. Each register group has a single one-bit input and output, the output being used for expansion purposes.

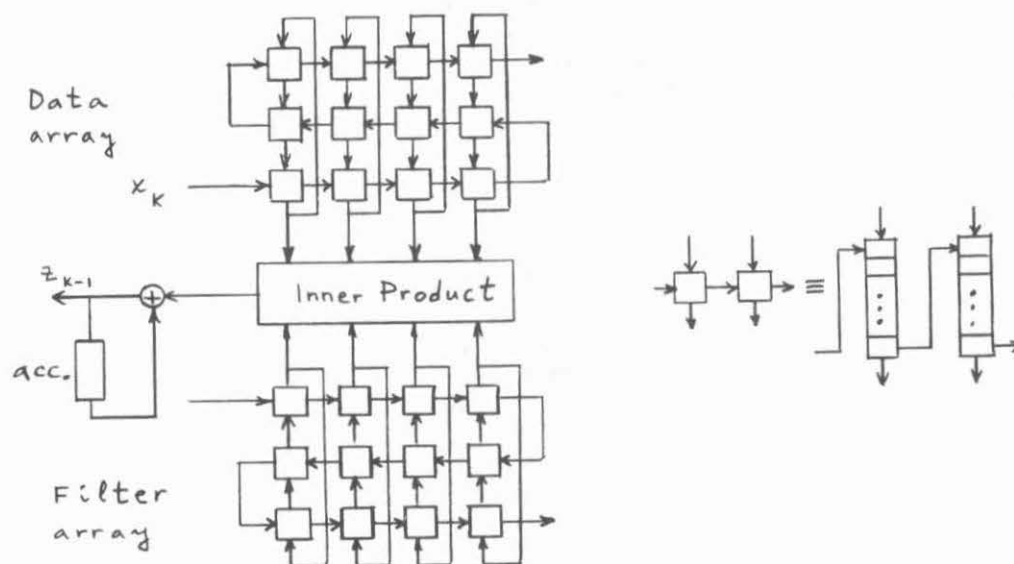


Fig. 7

The filter coefficients are loaded into the registers one at the time by using horizontal shifting. During each sample period the computation consists of 32 partial product cycles, and one memory shift cycle. Each partial product cycle results in a 16 component inner product, during which the registers are being shifted vertically. This provides bit-serial operand streams to the inner product processor, and simultaneously prepares new operands for the next cycle. At the same time the accumulator adds previous partial product to the one being computed. The last partial product cycle produces the result that is then shifted outside. In the next step a memory shift is done by shifting registers horizontally. The first register receives a new sample from the external source while it is transferring its content to the next neighbor to the right. At the same time the accumulator is cleared for the next round of partial product cycles. This sequence of steps is repeated for each new input sample.

This example is indicative of tradeoffs that have to be made in a practical design of array schemes for vector processing. The goal is to minimize the silicon area, while matching the processing speed with the available input/output data rates. Here, the area of two-dimensional shift register arrays was much smaller than an array of 512 inner product cells that could be used for the same computation. However, if the input sampling rate was on the order of 3 microseconds and the filter was specified with the same number of points, then a large processor array would be used. In addition, if the input rate was even larger, two arrays would have to be used, each operating on alternate samples.

A two-dimensional shift register array may be used in a similar way for other vector and matrix operations. An alternative to this approach is to use standard memory arrays with a specialized memory access facilities. An example is given in [9].

#### 4. The Multiplier

There are a number of reported bit-serial multipliers in literature [4], [5], [6]. Most of them preserve only  $N$  most significant bits of the result. We have devised yet another configuration, which preserves all  $2N-1$  bits. This is important for inner product computations, where a large number of individual products are accumulated.

If two integers are given in a binary representation, then they can be viewed as vectors whose components are in the set  $[0,1]$ . Then, their product is a vector whose elements are obtained by a convolution of the two operand vectors. Alternatively, a polynomial representation with a delay variable  $b^{-1}$ , can be used for representing integers for bit-serial arithmetic. A convolution of two vectors is equivalent to a polynomial multiplication, if the vector elements are equated with the polynomial coefficients. Let  $x$  and  $y$  be two  $N$ -bit integers, represented as

$$x = \sum_{i=0}^{N-1} x_i b^{-i}, \quad y = \sum_{i=0}^{N-1} y_i b^{-i}$$

The product polynomial is given by

$$z = \sum_{k=0}^{2N-2} z_k b^{-k}, \quad z_k = \sum_{i=0}^k x_i y_{k-i}$$

It is interesting to note that each  $z_k$  is an inner product of two binary vectors, so that a multiplier design becomes an exercise in configuring a regular array structure for inner product computation. In order to derive such a structure, we rewrite the result polynomial as:

$$z = \sum_{k=0}^{2N-2} (z'_k + z''_k) b^{-k}$$

$$z'_k = \sum_{i=0}^{\lfloor \frac{k}{2} \rfloor} x_i y_{k-i}, \quad z''_k = \sum_{i=0}^{\lfloor \frac{k+1}{2} \rfloor - 1} y_i x_{k-i}$$

These two expressions can be written in an iterative way, such that they map into a linear pipelined array of  $N$  sections. The  $j$ -th section computes the following:

$$z'_j = x_j y_k + z'_{j+1} b^{-1}, \quad j \leq k$$

$$z''_j = y_j x_k + z''_{j+1} b^{-1}, \quad j < k$$

$$k = 0, 1, \dots, N-1, \quad j = 0, 1, \dots, N-1, \quad z_j = z'_j + z''_j$$

The section 0 provides the product polynomial, with  $z_0 b^{-1}$  being the least significant bit. Notice that the additions in the above expressions are arithmetic, with a carry bit.

A diagram of a single section of the multiplier, and the connection of the sections, are shown in Figure 8. The operands are applied on two single bit buses,  $x$  and  $y$ , one bit per cycle. The control bit is provided simultaneously with the LS bits, and it advances from the 0-th section to the remaining stages synchronously with the bit rate. Its purpose is to enable  $x$  and  $y$  latches in each section. In the  $i$ -th cycle, it deposits  $i$ -th  $x$  and  $y$  bits in the  $i$ -th latches. Each section computes two partial sums,  $z'_i$  and  $z''_i$ . Carry-save adders add together three values, a product of two bit values, previous carry bit, and the delayed partial sum from the next section. The 0-th section also contains an adder for forming the final result. Finally, two's complement multiplication is obtained by applying a special "subtract" signal, simultaneously with the most significant bits of the operands,  $x_{N-1}$  and  $y_{N-1}$ . This has the effect of converting all adders to borrow-save subtractors at this time, (except in the last section). In this implementation, it takes  $2N-1$  steps to perform a multiplication of two  $N$  bit numbers.

The floor-plan of a single multiplier section looks very much like the diagram in Figure 8. All signals were chosen to run horizontally, so that a multiplier with an arbitrary number of bits can be constructed in many ways, by abutting sections on two edges only.



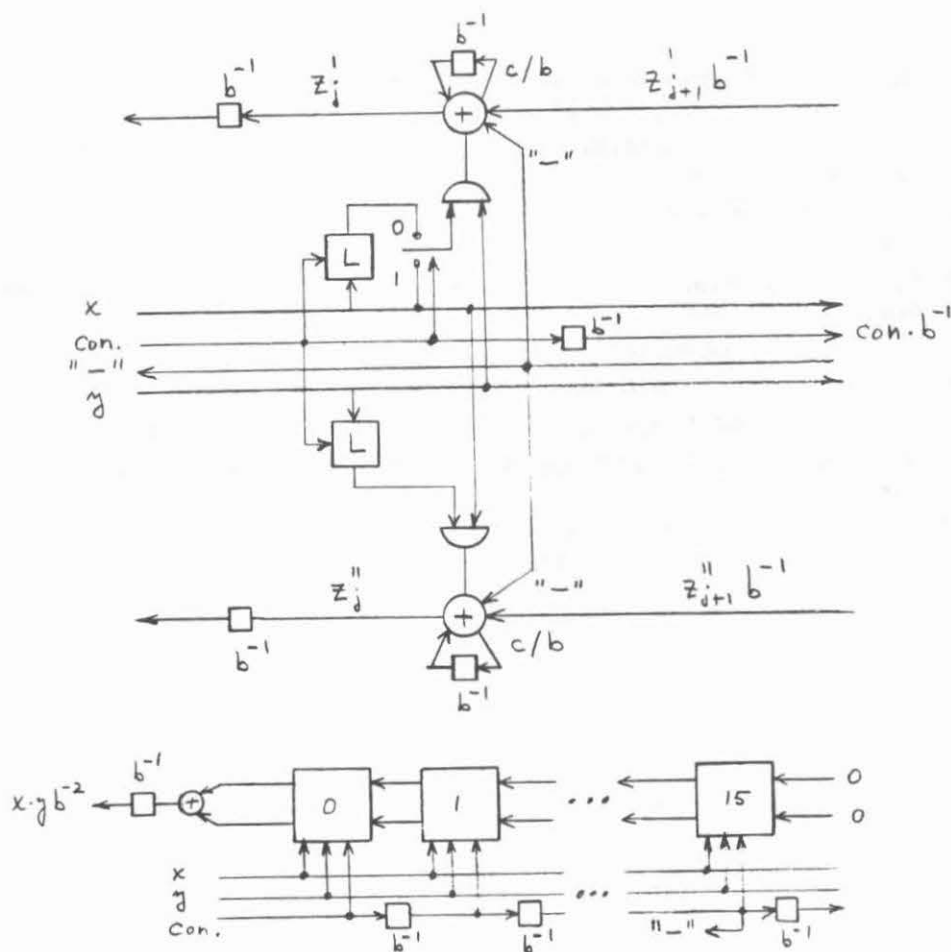


Fig. 8

## 5. Conclusion

A bit-serial approach to computations of vector inner products offers many advantages over word arithmetic. The size of basic processing elements and communication links is much smaller, and the array configurations are easy to implement. The slower rate of operation of a single multiplier-adder combination is offset by a much higher throughput rate of a large number of processors. A single element has been designed and tested, and a sixteen processor combination with a tree of adders is under way. The approach is especially useful for real-time signal processing tasks.

## 6. Acknowledgements

We are grateful to David Hagelbarger for suggesting the structure of the multiplier. Also, we would like to thank Sandy Fraser and Mike Maul for making the chip production possible.

## 7. References

1. C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980, 263-330.

2. H. T. Kung, *The Structure of Parallel Algorithms*, Carnegie-Mellon University, August 1979.
3. K. E. Batcher, "Design of a Massively Parallel Processor", *IEEE Trans. Comput.*, Vol. C-29, pp. 836-840, Sept. 1980.
4. E. K. Cheng and C. A. Mead, "A Two's Complement Pipeline Multiplier", *Proc. ICASSP*, Apr. 1976.
5. R. F. Lyon, "Two's Complement Pipeline Multipliers" *IEEE Trans. Commun.*, 418-425, Apr. 1976.
6. L. B. Jackson et al., "An Approach to the Implementation of Digital Filters", *IEEE Trans. Audio Electroacoust.*, vol. AU-16, pp. 413-421, Sept. 1968.
7. Special Issue on Parallel Processing, *IEEE Trans. Comput.*, vol. C-29, Sept. 1980.
8. E. E. Swartzlander et al., "Inner Product Computers", *IEEE Trans. Comput.*, vol. C-27, pp. 21-31, Jan. 1978.
9. K. E. Batcher, "The Multidimensional Access Memory in STARAN", *IEEE Trans. Comput.*, vol. C-26, pp. 174-177, Feb. 1977.

## **A SMART MEMORY ARRAY PROCESSOR FOR TWO LAYER PATH FINDING\***

**Christopher R. Carroll, Caltech**

This paper describes three examples of hardware implementations of path finding schemes based on the Lee-Moore maze solving algorithm. One is purely a demonstration circuit to show the technique. The other two are complete LSI implementations which should be usable in building large and useful path finding machines. One of these two LSI circuits, known as the MAZER, is designed to find shortest paths from one point to another on a plane, where there is only *one* layer of allowable routes to take. As its name suggests, this chip solves ordinary mazes, or on a more practical level, it can route wires on a one sided printed circuit board. The other LSI circuit, known as the PATHFINDER, is designed to handle the two sided printed circuit board case. It finds a *least costly* path from one point to another where there are *two* parallel planes on which routes are allowed. Crossing of the path from one plane to another can be either unrestricted, as in *free via* printed circuit boards, or permitted only in certain places, as in *fixed via* boards. The phrase "least costly" above can, for now, be read as "shortest", although in a later section a more general definition will be revealed.

The remainder of this document is divided into three parts. The first section outlines the original *Lee-Moore algorithm* for path finding, on which the circuits described later are based. The second section details the *one layer hardware*, including both the *demonstration circuit* and the *MAZER* chip. Finally, the third section describes the *PATHFINDER* chip and the techniques used to conquer the problems encountered in two layer path finding. Documentation on the integrated circuits includes those results of testing and characterization which were available at the time of this writing.

---

\*The research described in this paper was sponsored by the Defense Advanced Research Projects Agency, ARPA Order number 3771, and monitored by the Office of Naval Research under contract number N00014-79-C-0597.

## 1. THE LEE-MOORE ALGORITHM

The *Lee-Moore algorithm* for path finding, proposed by E. Moore in 1959 (1) and extended by C. Lee in 1961 (2), is a scheme for finding the shortest route between two points in a plane, where the route is composed of some number of vertical and horizontal segments through a rectangular grid superimposed on the plane.\* This has been a popular algorithm for people doing problems related to maze solving because it is easy to implement and because it guarantees that a path will be found if one exists. The drawbacks to the algorithm are that it is expensive computationally in both time and space. However, the use of the hardware described in this paper circumvents these difficulties.

Suppose that the size of the grid, i.e. the pitch of the cells defined by the grid, is set to the minimum path width that is allowed. In the case of printed circuit board design this would be the minimum center to center spacing for adjacent wires. Suppose also that the grid is uniform and symmetric, forming an array of square cells, each a path width on a side. The path found by the algorithm from point A to point B will consist of a route beginning at the cell containing point A, continuing to a neighbor of that cell, and then to a neighbor of that second cell, and so on from a cell to one of its neighbors, until eventually the path ends in the cell containing point B. Some of the cells in the array may be *blocked*, preventing the path from running through these cells. These would be "barriers", or "walls" in a maze, or cells occupied by previously routed wires in the printed circuit board application.

The algorithm finds the shortest path from A to B in two phases. One word of storage, which I will call the "label", is associated with each of the cells in the array. The first phase, called the Propagation Phase, stores information in the labels throughout the array. The second phase, called the Retrace Phase, then uses that information to find the required path.

---

\* Since our geometry here is based on this grid, the distances mentioned will be *Manhattan distances*, i.e. the distance from A to B would be the shortest distance covered by a taxi driver driving from A to B on the streets of Manhattan.

The Propagation Phase, which distributes the information, executes the following program:

```

put label -1 in all cells which are blocked
put label 0 in all cells which are not blocked
N:=1
put label N in the cell containing point A
while cell containing point B is labelled 0 and more activity is possible do
begin
  for every neighbor of every cell labelled N do
    if that neighbor is labelled 0 then label it (N+1) else leave it alone.
  N:=N+1
end

```

This part of the algorithm is illustrated in Figure 1. The purpose of this phase is to distribute information to the cells which can then be used to find the direction back to point A. The information is spread out in a propagating wavefront centered on point A, much like waves propagating away from a stone dropped in a pond. It is interesting to note that the only activity that takes place occurs at the frontier of this expanding wavefront. Cells ahead of the frontier merely wait for the wave to arrive, keeping their label of 0. Cells behind the frontier have already received the information they need, and simply keep it stored in their label.

The Retrace Phase, using the information stored in the labels, executes the following program to find the path:

```

Start the path at the cell containing point B
N:= label of cell containing point B
if N=0 then there is no path from point A to point B
else begin
  While path has not yet reached cell containing point A do
  begin
    N:=N-1
    Continue the path to a neighbor of the current cell which contains
    the label N
  end
end

```

This part of the algorithm is illustrated in Figure 2. Notice that there is nothing to specify which cell to choose when there are two or more possible choices. This merely means that there are multiple paths between A and B that have the same length. To first order, there is thus no preference of one path over another, so no selection mechanism need be used. In practice, some scheme is often employed when there is a choice of paths to take. A common selection scheme is to avoid changing

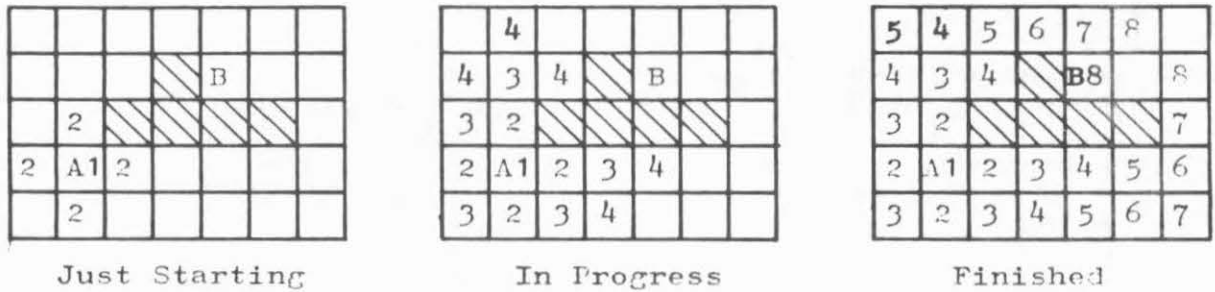


Figure 1. The Lee-Moore Propagation Phase

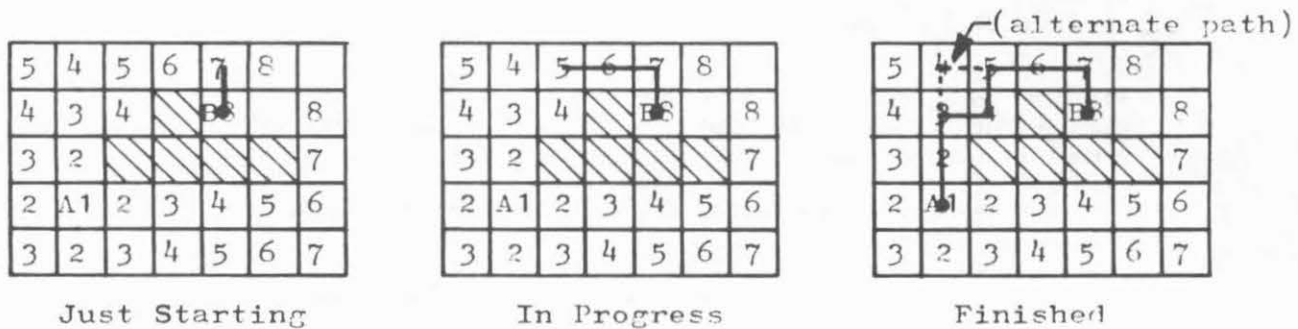


Figure 2. The Lee-Moore Retrace Phase

directions in the path during the Retrace Phase when it is unnecessary. This tends to minimize the number of bends in the resulting path. When this phase is complete, the algorithm either has found the required path from A to B or has proven that no such path exists.

Before beginning the discussion of the hardware implementations of this algorithm, a couple of things should be noted. First, examine the time complexity of the programs above. The Retrace Phase merely traces the path from B back to A using information stored in the cells. The only cells accessed are those along the selected path and their immediate neighbors. The time complexity is thus *linear* with respect to the path length. However, in the Propagation Phase, the situation is worse. Information is propagated in all directions around point A. The number of cells accessed is approximately proportional to the square of the path length. Thus, the time complexity here is *quadratic* with respect to the path length, making the algorithm as a whole quadratic. This is unfortunate, since for maze solving to be interesting, a large maze must be involved. In the circuit board application, for example, a cell array containing 1000 x 1000 cells would be common. The quadratic time aspect of the algorithm thus is a real handicap. Current software using this algorithm to route typical printed circuit boards can consume several hours of CPU time on a full-size computer. On top of that, the space requirement is also large. Circuit board routing requires 10-12 bits of storage for each cell, and a million 12 bit words is a lot of memory. So, both the space and time complexity of the algorithm need to be attacked in any successful hardware implementation. The next section shows how this was accomplished.

## 2. HARDWARE FOR FINDING ONE-LAYER PATHS

Implementing the Lee-Moore algorithm in hardware is a clean and natural thing to do. Because the problem is cellular and because information flows only between adjacent cells without using any long distance communication paths, the task is a natural one for an array processor structure with one processor per cell in the array. However, if there is to be any hope of building a large machine this way, there are two problems which must be overcome. First, *the amount of storage per cell* must be limited. In the original algorithm, in an array of unbounded size each cell would be required to contain an unbounded number of bits. Second, the *global state* required in the original algorithm, which was represented by "N" in the programs above, must be eliminated. Accomplishing these goals would result in a machine which could be extended to any size needed without undue complications.

The first goal, that of limiting the amount of storage required in each cell, was attacked by S. Akers in 1967 (3). He showed that only two bits were required per cell to implement the algorithm proposed by Lee and Moore. Of the four states available from the two bits, one indicated that a cell was blocked and unavailable for new paths. Another was used to indicate a cell that was so far untouched by the propagation process, like label 0 in the above programs. Then, instead of using the ascending ordinal numbers to label successive wavefronts in the propagation, Akers used successive members from the sequence

1, 1, 2, 2, 1, 1, 2, 2, 1, 1, 2, 2, . . .

These last two states stored the information necessary to get back to the point where the propagation started. See Figure 3. It was only required that the program remember whether it was on the first 1, second 1, first 2, or second 2 in the sequence when it stored a number in the goal cell containing point B. For example, if the program knew it was on the second 1 of a pair when it reached the goal, then in the Retrace Phase it looked for cells containing the above sequence in reverse order, starting with the first 1, then 2,2,1,1, etc., until it reached the starting cell. This was a big step forward, for only two bits of storage were needed for each cell, no matter how big the array of cells was made. Unfortunately this did nothing to solve the problem of having to distribute the next member of the sequence as a global variable to all the cells in the array.

The solution to the dilemma of global state is to use a slightly different strategy in the algorithm. This idea was recorded in an internal document of the Caltech Computer Science Department by Ivan Sutherland (4). Rather than numbering successive wavefronts with some sequence and then searching for the reverse of that sequence to find the path, simply store in each cell arrows which point to the neighbor(s) from which the wavefront approaches as it passes over that cell, and then just let the arrows show the path back to the starting cell. This approach is shown in Figure 4. Since the wavefront may reach a cell from more than one neighbor simultaneously, and since that fact is important when trying to select one of several equally good paths, an arrow for each neighbor is needed. These arrows require one bit each, because the wavefront either came from that neighbor or it didn't. Additionally, one bit is required to identify a cell as being blocked. Five bits per cell is more than Akers' two, but it is still a small number, and more importantly, it is still a



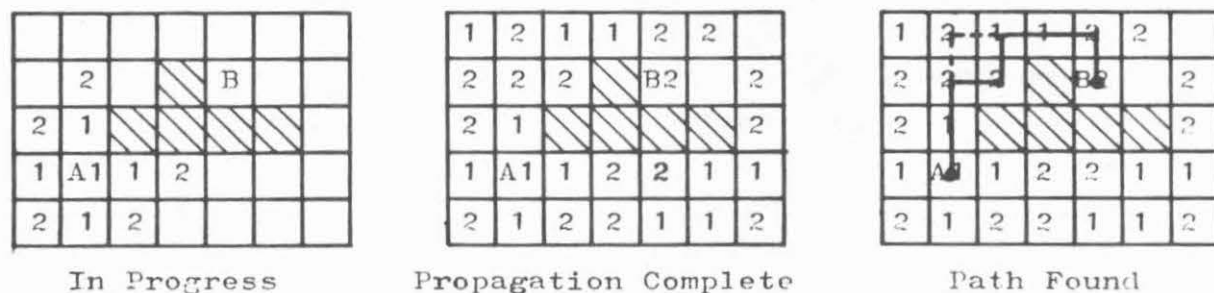


Figure 3. Akers' Modification

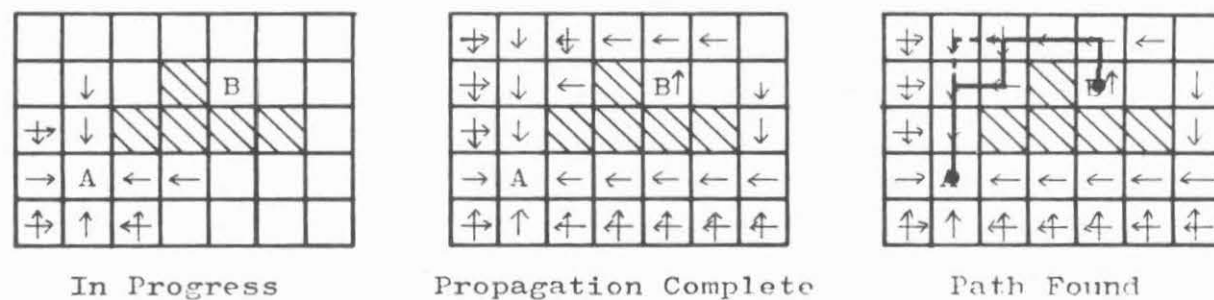


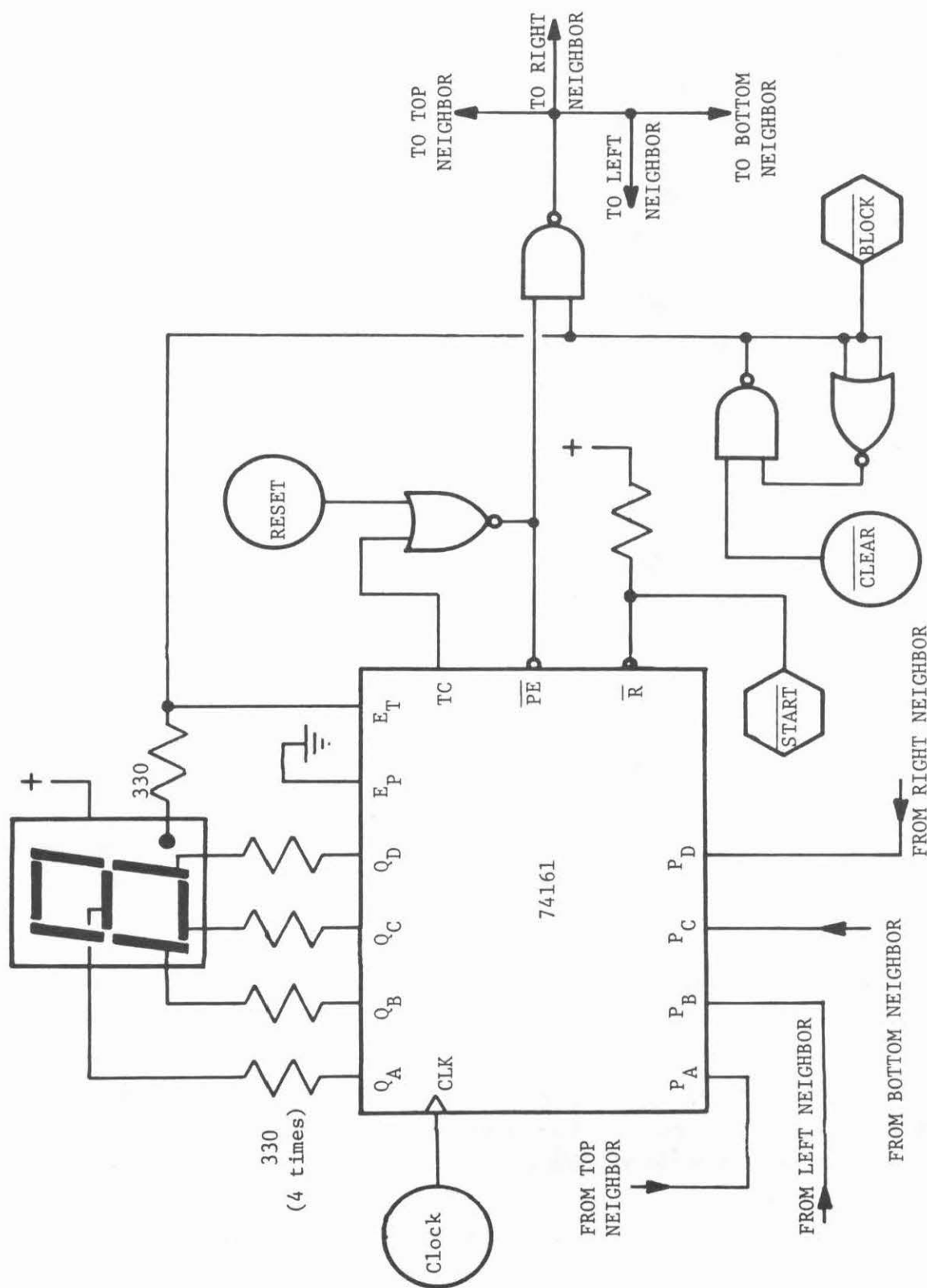
Figure 4. Modification for MAZER

bounded number, that does not change as the array of cells grows. With these modifications to the algorithm, the move to hardware is hardly more than just "wiring it up".

## 2.1 THE DEMONSTRATION CIRCUIT

As a demonstration of the feasibility of this approach to path finding, I built a small array of processors out of standard TTL parts. Figure 5 shows the circuit I used. As can be seen from the figure, there is not much to the "processor". It consists of one and two halves standard TTL packages, a few resistors, and an LED display. The circuit uses a 74161 as a four bit latch. The 74161 features the TC output, which is the logical AND of the four latch outputs and the  $E_T$  input. The four bits in the 74161 are the four arrow bits. The fifth bit is formed by one NAND and one NOR gate to indicate the blocked condition. Global control signals are circled. The two signals START and BLOCK are independent for each cell and are activated by momentarily grounding that node with a probe tip. Communication with neighbor processors enters this processor at the preset inputs of the 74161 and exits to the neighbors from the NAND gate at the right.

In operation, the circuit is quite simple. Initially, the CLEAR signal is taken low to clear all the block flip-flops. Then the maze walls are defined by selectively blocking some processors by grounding their BLOCK inputs. The LED decimal point lights in the blocked cells. Next, RESET is taken high for at least one clock cycle. This forces all communication wires between neighbor processors high and parallel loads all ones into the 74161, turning off the LED segments. This is a stable configuration, and will not change as the clock ticks. All communication wires stay high, and the latches keep parallel loading all ones because the TC outputs are high. Now suppose that somehow the processor to the right of this one changes out of its all ones state. Then its TC output goes low, causing the latch to stop parallel loading and causing the communication wires leaving the processor to go low. One of those wires enters this processor on the  $P_D$  input. At the next clock cycle, that low state is loaded into the D bit of the latch, turns on the right LED segment indicating a right pointing arrow, and prevents further parallel loading of the latch by forcing the TC output low. The result, then, is an indication on the LED for this processor that something happened to the right of it. Incidentally, when the TC output of this processor went low, so did the outgoing communication wires, so on the next clock cycle, the other neighbors of this processor will be activated just as described above. Now, how did all that get



started? Well, the START input on one cell was momentarily grounded, causing the latch outputs to go to all zero, turning on all four LED segments indicating that propagation started there, and causing that cell's communication outputs to go low. It is actually a very simple process that each processor in the array must execute. There is no computation in the numerical sense involved. Each cell simply passes on the propagating signal when it arrives, and records from which direction(s) it came.

When the propagation reaches the edges of the array, or can go no farther because of blocked cells, the action stops. What is recorded by the LEDs is actually the direction to go from each cell in the array to get back to the cell where it all started. The hardware has found the shortest path from the starting point to any other point in the array.

I designed this circuit purely for demonstration purposes. As such, tracing the path back is a visual process done by looking at the LED displays. If automatic trace back were desired, the five bits in each processor would be accessed as five bit words by a general computer which would then consider the processor array as a block of smart memory. It is an easy task for an ordinary computer to decipher the bits from each processor to find the path desired.

Before proceeding, consider what has happened to the computational effort required to reach this result. Time complexity of the algorithm has been dramatically improved. Now, rather than having a *single processor* advance the wavefront by stepping around the starting cell one cell at a time in an expanding spiral, the propagation takes place by activation of successive *rings of processors* surrounding the starting cell. At any given time, a number of processors directly proportional to the length of the path are actively working, rather than just one processor. The time required for the wavefront to expand out to the goal point is now directly proportional to the length of the path, not to the square of the length. Thus, the time complexity of the algorithm is now *linear*, not quadratic, with respect to the path length. This result is expected -- a linear number of active processors can do in linear time what one active processor can do in quadratic time.

The circuit described above is so simple that it seems natural to lay out several copies of it on a silicon chip. That is just what was done for the design of the *MAZER* chip.

## 2.2 THE MAZER CHIP

The first step in designing the *MAZER* chip was to develop a NMOS circuit which performed the function of the demonstration circuit above. The stumbling block was the clocking scheme. Edge triggered latches are not as easy to come by in MOS as they are in TTL. Usually a set of multi-phase clocks are used to latch signals. This seemed to unnecessarily complicate the circuit, and a way around the problem was sought. The answer turned out to be easy. Just don't use any clocks!

On careful scrutiny of the operation of the demonstration circuit, one sees that clocking is really unnecessary. The only operation performed by each processor consists of waiting for the propagating wavefront to reach it, recording the direction(s) from which it came, and passing it along to its neighbors. One could imagine the array of processors as an array of mousetraps, each cocked and ready to fire. Each mousetrap is designed to fire as soon as any of its neighbors fire. Each mousetrap will store the direction from which its firing signal comes. At the end of the outward propagation process, which might always be allowed to propagate to the extremities of the array, the contents of each mousetrap cell's storage would then be the required arrows pointing in the direction of the shortest path from that cell to the start of the wave propagation process. This is a good visualization of the way in which the *MAZER* works.

Figure 6 is a conceptual logic design of a simplified *MAZER* cell. Not shown are all the mechanisms for accessing the cell, blocking the cell so that it becomes part of a "wall" in the maze, causing that cell to be the starting point of wave propagation, etc., but the mousetrap characteristic is illustrated. After the reset line has gone high to make all the flip flop Q outputs low, all signals which cross the cell boundary are low, and the system is stable in this state. Now, if for some reason one of the incoming signals goes high, the corresponding flip flop will be set. This causes the inputs to be disabled via the AND gates, and also causes the cell to generate a high going signal to each of its neighbors, triggering them in the same way. The flip flops remember from which direction the activation signal entered the cell, and reading them out by an accessing mechanism not shown gives the direction the maze solution takes as it passes through this cell.

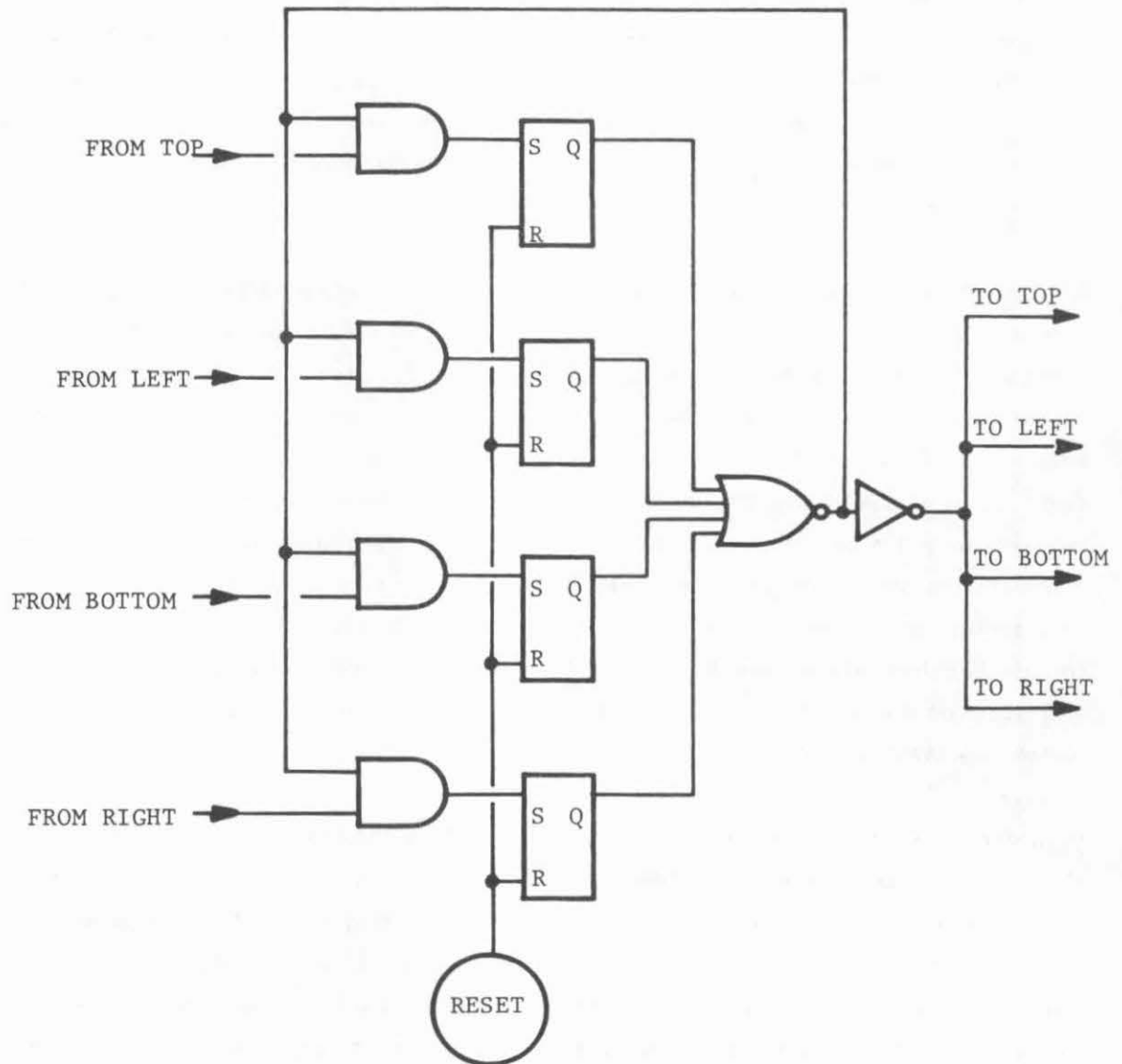


Figure 6. The Mousetrap Concept

Figure 7 is an actual schematic of a *MAZER* cell. Three of the AND gate/flip flop combinations of Figure 6 are seen here as transistor groups Q1-Q6, Q7-Q12, and Q13-Q18. The fourth direction is identified by the state where the cell has been triggered, and the other three flip flops are not set. The NOR gate and inverter are formed by Q19-Q25. Four bits of information are provided, as open drain outputs wire OR-ed with other cells on the chip. These bits are the three flip flop outputs plus a signal which indicates if the cell mousetrap has been "sprung". Transistors Q33-Q36 form a flip flop to store the blocked condition. ROW and COLUMN are addressing signals to select the cell for data readout, BLOCKing the cell to make it part of a maze wall, or STARTing the propagation process with this cell. RESET re-cocks the mousetraps, but does not destroy the blocked condition in the maze wall cells. CLEAR unblocks all the cells in preparation for a new maze. The other signals are communication paths to adjacent cells.

The complete *MAZER* chip contains sixteen processors arranged in a four by four array. Larger arrays can be assembled by arranging *MAZER* chips themselves in an array. Four wires come off each edge of the chip for the purpose of communication to adjacent chips. There are 15 additional wires which come off chip for data and control. Four are for data outputs, four are for address inputs, two are for power, and five are for the control signals BLOCK, RESET, CLEAR, START, and CHIP-ENABLE. A plot of the *MAZER* chip is shown in Figure 8. The four by four array of processor cells can be seen, surrounded by the 31 pads. Designed with Caltech's conservative design rules, the chip measures 2241 microns square, or about 8100 square mils.

### 2.3 MAZER CHIP TEST AND CHARACTERIZATION

The *MAZER* chip recently returned from fabrication and some preliminary results of testing are in. I have three chips which apparently contain no processing faults. The chips seemed to perform strangely, until I diagnosed the symptoms and found a basic bug in the *MAZER* processor circuitry. With the bug uncovered, I was able to circumvent the problem and continue to test the parts.

In order to understand the bug in the *MAZER* circuitry, examine Figure 7. The four data outputs of each of the sixteen processors, which come from the drains of transistors Q27 through Q30 in the figure, are bussed together onto four global data wires on the chip, DATA1 through DATA4. These wires run to the chip output pad circuitry, where there is a single pull up transistor on each of them. The important

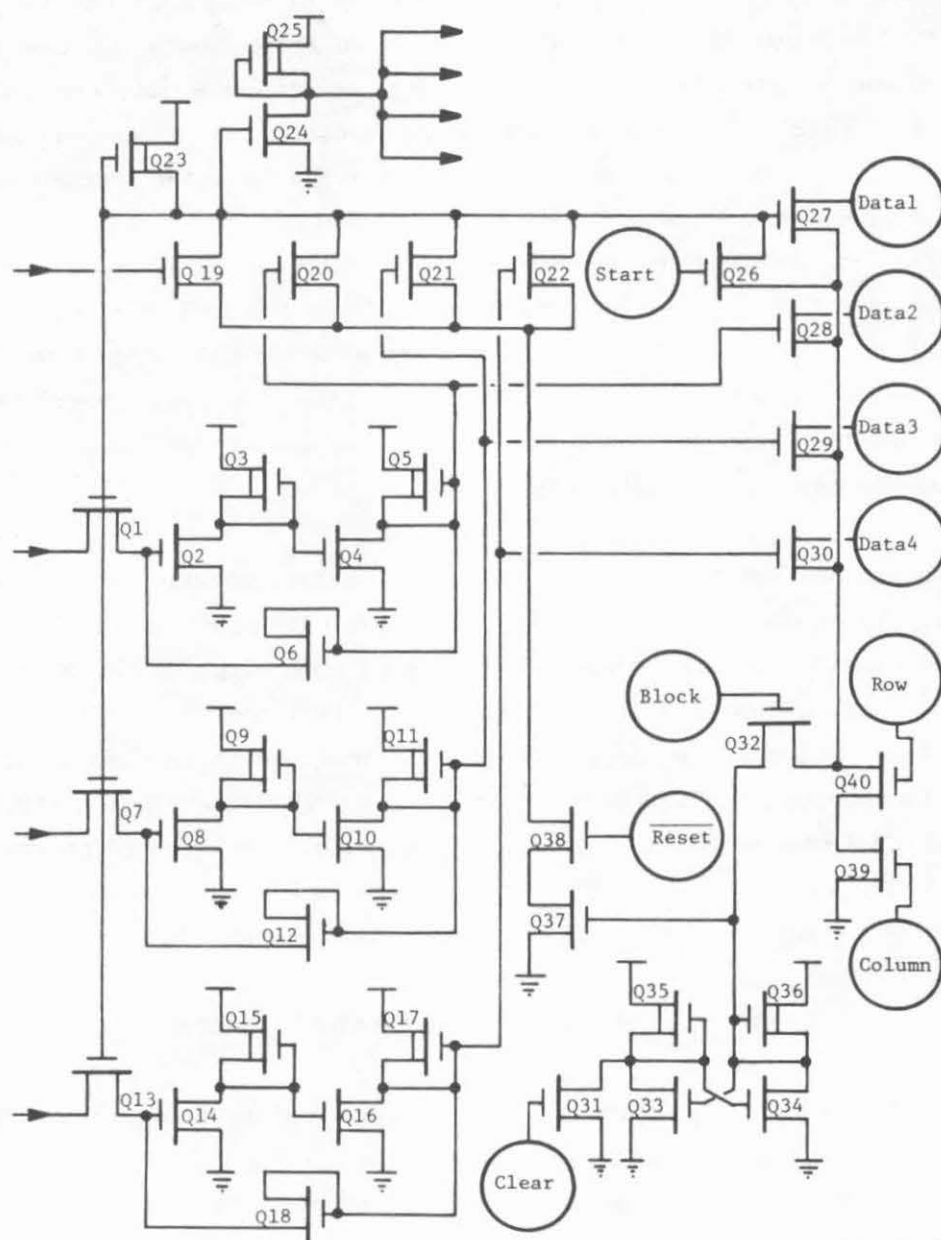


Figure 7. Schematic of a MAZER cell



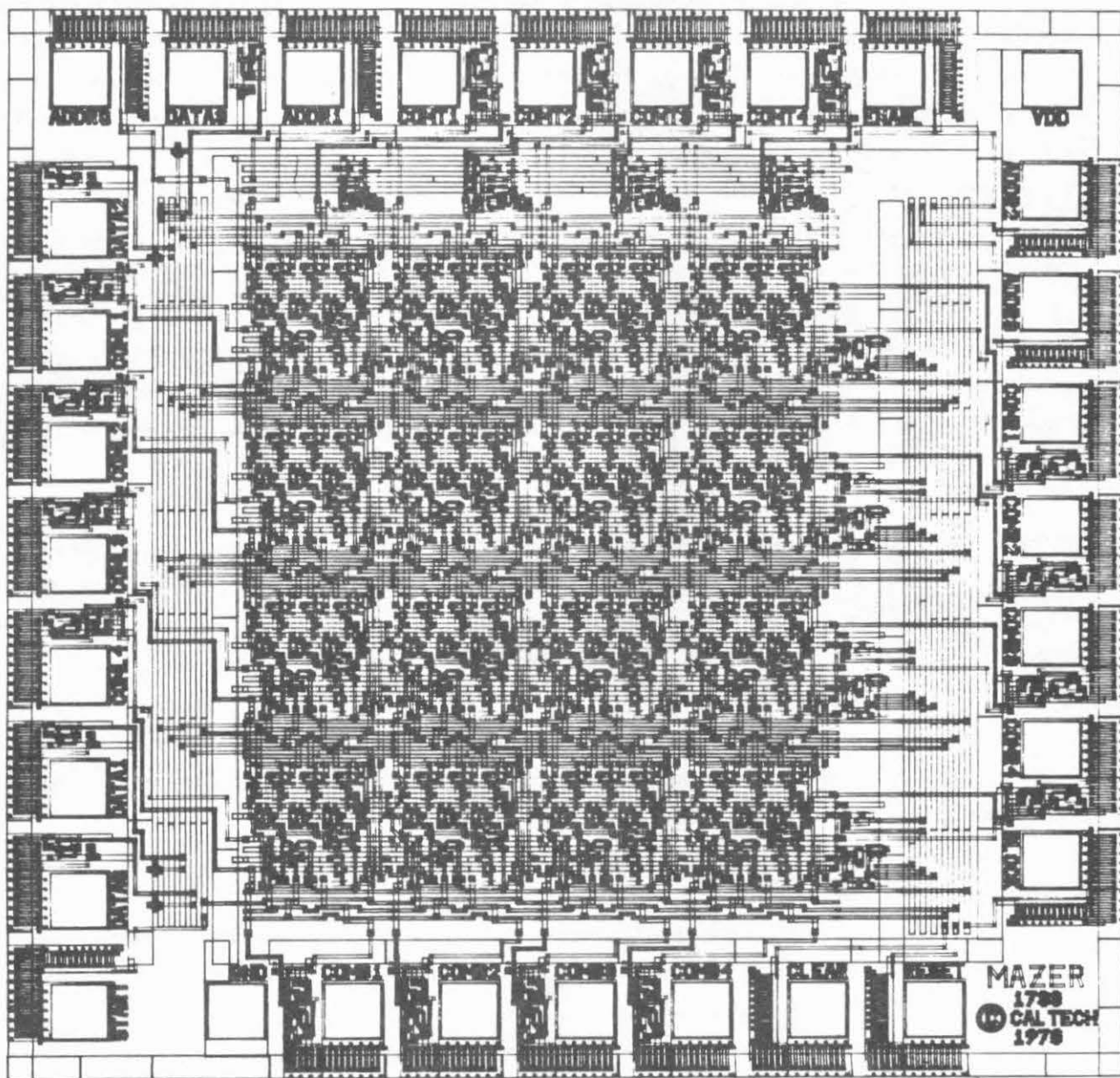


Figure 8. Plot of the MAZER chip

point to examine is the node in each processor which is common to the sources of the four transistors, Q27 through Q30. This node, which I will call the "enable node", is pulled low by the decoding transistors, Q39 and Q40, which are controlled by the addressing signals ROW and COLUMN. Since Q39 and Q40 are both turned on in only one of the sixteen processors, there should be a path from the enable node to ground only in that addressed processor. This prevents data in the non-addressed processors from pulling down on the data lines. However, things are not so simple. Suppose one of the data output transistors, say Q27, is on in the addressed processor, thus pulling down the DATA1 wire. Now, suppose that in another processor, which is not being addressed, both Q27 and Q28 are on. Since DATA1 is being held low by data in the addressed processor, Q27 in the second processor provides a path to ground for the enable node in that second processor. As a result, Q28 in that second processor can erroneously pull down the DATA2 wire. Since the addressed processor should not pull down DATA2, the result is that incorrect data appears at the output of the chip.

Fortunately, it is possible to retrieve correct data despite that problem. In the above example, notice that DATA1 is pulled down first by the addressed processor in the normal way. The bad data does not start to pull down on DATA2 until DATA1 is down, and even then, the string of transistors doing the pulling on DATA2 is longer than normal. The result is that good data shows up on the chip output pads about fifty nano-seconds before it is ruined by bad data caused by the sneak paths. Latching the good data in an external latch at the right time retrieves the correct state of the internal bits.

The operations of STARTing propagation and BLOCKing the cell are also affected by the unwanted paths between enable nodes and ground. Luckily, because of the high pull-up to pull-down ratio in the Q24/Q25 inverter, resulting in a very low switching threshold, STARTing can be performed normally. Apparently the sneak currents are low enough to prevent the inverter from switching in the non-addressed processors. However, so far I have been unable to individually BLOCK cells. The effect of blocked cells can nevertheless be tested by using the random distribution of blocked cells present after power-on. The CLEAR signal, which clears all blocked cells, operates normally.

In spite of the difficulties mentioned above, the test results are encouraging. All the arrows point correctly back to the cell where propagation starts. Some local

asymmetries sometimes are present, indicating that propagation proceeds a little more quickly in some areas of the chip than in others, but this result is expected with the asynchronous scheme used, and is negligible anyway. Access time, from chip enable to data output, is around 100 nano-seconds, about normal for a chip of this small size. A more detailed characterization of the chip remains to be completed.

### 3. HARDWARE FOR FINDING TWO-LAYER PATHS

The fact that the *MAZER* is limited to single layer paths limits its usefulness. The most immediate application for path finding hardware is in the area of printed circuit board design. However, single sided circuit boards are not very exciting. The step to two sided boards dramatically improves wireability and board density. Adding even more layers to the board improves density still more, but the additional effort does not buy nearly as much as the move from one to two sides. Thus, there was great incentive to develop a two layer path finder, with the specific goal of producing a routing machine for two sided printed circuit boards. This is the background for the design of the other integrated circuit to be discussed, the *PATHFINDER*.

At first glance, it would seem that one need only construct a circuit that forms the topology of two *MAZER* chips laid on top of one another, with an additional arrow bit in each cell to indicate travel from one layer to the other. This strategy would work, except that it lacks some properties which have been found very desirable in the two layer environment. In what follows, terminology of the printed circuit board world will be used, with the understanding that other applications, such as interconnect wiring on integrated circuits, would have analogous features and terminology.

The first feature that would be missing in such a two-layer *MAZER* is the ability to block travel from one side of the board to the other independently from blocking travel through those cells without changing sides. Often it is desirable to prevent these holes in the board, or vias, from occurring in certain areas of the circuit board. Perhaps vias are to allowed only on a tenth inch grid, for example. Furthermore, vias sometimes can affect more than just the cell in which they occur. A via in one cell may prohibit the placing of a via in an adjacent cell. For all these reasons, an additional bit is required for via blocking in any proposed two layer path finding system to make it useful.

The second missing feature is much more disturbing. Designers of two layer circuit boards have long realized that it was advantageous to employ a tendency for wire runs that were mostly vertical to end up on one side of the board, and runs that were mostly horizontal to end up on the other side. This helps to avoid unnecessarily blocking channels for future wires. The tendency of a wire to choose one side of the board or the other depending on its orientation would be completely lacking in a straightforward two-layer *MAZER*. Incorporating this preference into the basic path finding algorithm was an interesting problem, and the methods developed to solve it in both the traditional software implementations and the current hardware implementation will now be examined.

A way to achieve the wire location preference is to use a system of *costs* associated with travel from cell to cell through the array. One group to incorporate these costs into a standard software wire routing system was a group at Burroughs Corporation (5). Each cell stored one integer, but rather than storing ascending ordinal numbers on successive wavefronts in the propagation phase, as in the original algorithm, each cell stored the accumulated cost for reaching that cell from the starting cell. Suppose  $C(a,b)$  is the cost for expanding the wavefront from cell  $a$  to its neighbor, cell  $b$ . Then as the wavefront passes from cell  $a$  to cell  $b$ , the number stored in cell  $b$  is the number stored in cell  $a$  plus  $C(a,b)$ . Since different costs might be encountered along different routes from the starting point to a given goal point, a number which has been previously stored in a cell might be overwritten if the wavefront reaches that cell from another direction with a lower cost than that achieved by the first contact with the cell. This is shown in Figure 9. Notice that the wavefront expands in exactly the same way as it did in the original algorithm, but now cells on the frontier are not necessarily all equally "distant" in terms of costs, as they were in the schemes described earlier. In the retrace phase of the algorithm, the numbers stored in the cells are used in a way similar to that described earlier. However, rather than searching neighbor cells for the next member in a reversed sequence, each step of the retrace involves searching for a neighbor of the current cell with a stored cost less than that of the current cell by the amount of the cost of propagating from that neighbor to this cell during the propagation phase. This does not necessarily result in the shortest path from point  $A$  to point  $B$ . What comes out instead is the least costly path between those two points, based on the cost function  $C(a,b)$ . This is the meaning of the phrase "least costly" in the introductory paragraph of this paper.

	11					
2	A1	2				
	11					

Just Starting

	21					
12	11					
2	A1	2	3			
12	11	12				

A little more

32	31	32				
22	21	22				
12	11					
2	A1	2	3	4	5	
12	11	12	13	14		

Still Farther

32	31	32	33	34		
22	21	22		44		26
12	11					16
2	A1	2	3	4	5	6
12	11	12	13	14	15	16

Not Done Yet!

32	31	32	33	34	35	36
22	21	22		28	27	26
12	11					16
2	A1	2	3	4	5	6
12	11	12	13	14	15	16

Finished

32	31	32	33	34	35	36
22	21	22		28	27	26
12	11					16
2	A1	2	3	4	5	6
12	11	12	13	14	15	16

Path Found

Figure 9. Path found by using a cost of 1 for travel in the horizontal direction and 10 for travel in the vertical direction.

The simplest cost function normally used consists of only three distinct costs. One cost is used for travelling in the "easy" directions, north-south on one side of the board and east-west on the other side, a second, slightly higher, cost is used for travelling in the "hard" directions, east-west on the first side and north-south on the second side, and a third, even higher, cost is used for travel "through the board", from one side to the other. Fancier schemes are possible. These involve reduced costs for travel near to and parallel to the edges of the board to increase utilization of that area, increased costs near component pins to prevent blocking future access to those pins, etc. The task of developing a cost function tailor-made to a particular circuit board can become quite an art.

Note, however, that using this cost function as a solution to the two layer situation brought back the same problems the original algorithm had, namely an unbounded number of bits of storage per cell, and global distribution of a numerical cost function. If there was to be any hope of building a chip comparable to the *MAZER* for two layer circuit boards, these problems had to be eliminated. To accomplish this, the *MAZER* was re-examined.

The scheme of using arrows instead of numbers seemed to be the way to go to limit the number of bits per cell. Using this method, however, required that during the propagation phase the expanding frontier of the wavefront must include only cells that were equally distant in terms of cost from the starting point, unlike the above two layer approach. This seemed to be at odds with the uniform, diamond shaped wavefront propagation described above.

The solution to the costs problem was to control the speed of wavefront propagation from cell to cell, rather than let it go at gate delay speeds. Consider the simple three cost system described earlier. If propagation could be allowed to proceed quickly in the "easy" direction, more slowly in the "hard" direction, and even more slowly in the "through the board" direction, the wavefront would meet the requirement that all cells on the frontier would be at an equal "distance" in terms of cost from the starting cell. Imagine a system where north-south propagation is easy on the top of the board and hard on the bottom. On such a board, a wavefront propagating from point A to a point B directly north of point A will reach point B on top of the board first, and will thus store arrows indicating a path which travels on the top side of the board back to point A. Similarly, if point B were to the east of point A, the wavefront, propagating more quickly in the east direction on the bottom of the board than on the top, would



cause point B to store arrows indicating retrace along the bottom of the board back to point A. No matter where point B was, the arrival of the wavefront would store information describing the least costly path back to point A.

During the time this solution evolved, some redundancy in the storage used in the **MAZER** made itself known. If the propagation phase left an arrow in cell A pointing to a neighbor cell B, indicating that retrace should proceed in that direction, that implied that there would be no arrow in cell B which pointed to cell A. Since that particular combination, adjacent cells pointing at each other, would never occur, there must have been some redundant information stored there, implying that a reduction in storage was possible. This was accomplished by moving the location of the arrow bits from inside each cell to between cells. Since each arrow then served the two cells between which it lay, the total storage required for the arrows was halved.

With these new modifications to the basic Lee-Moore algorithm, it was time to start designing the two layer chip.

### 3.1 THE PATHFINDER CHIP

The difficult obstacle in the design of the **PATHFINDER** chip was the method to use in controlling propagation speeds. What was required was a way to vary the speed over at least a ten to one range in each of three directions, the "easy" direction, the "hard" direction, and the "through the board" direction. Also, the circuitry could not be overly complex, nor could it involve many wires to the global environment. However, the required speed settings were related to the cost function described above. The cost function was something that was set by little more than educated guessing and experimentation. There was nothing very critical about the exact values of the costs. Only approximate settings were required. All of these considerations led the design away from a digitally controlled speed system, and towards an analog system.

The method employed relies heavily on the dynamic charge storage abilities of MOS circuitry. Figure 10 shows the set up for a simplified, one layer cell with its surrounding arrows, not showing the blocking or accessing circuitry. Each cell contains a capacitor of about 5 pF, or so. Before the start of the propagation phase, the capacitors are all precharged by means of the precharge transistor. With all the capacitors charged, all the arrow flip flops have both outputs held low. To start

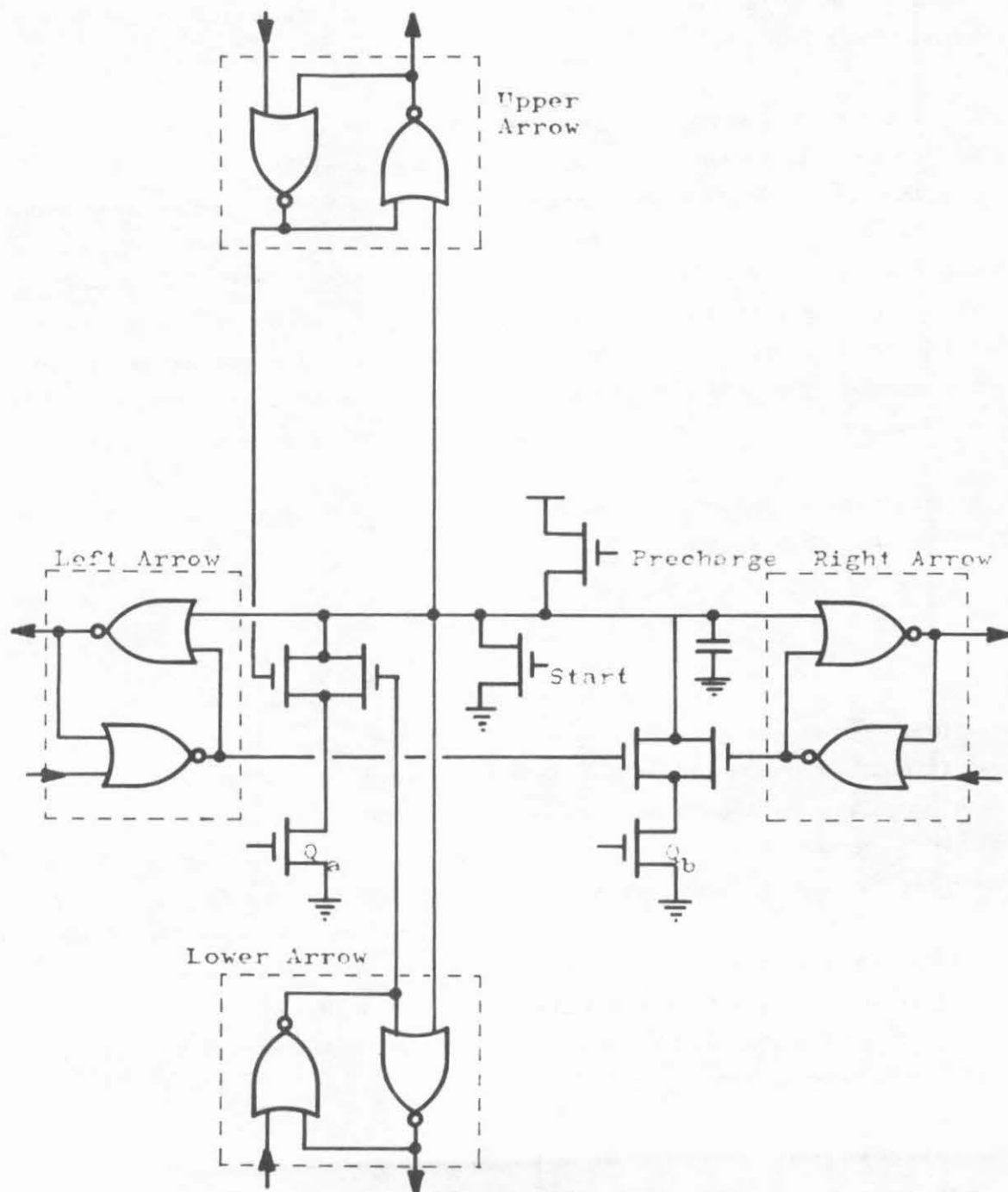


Figure 10. Simplified PATHFINDER Cell



propagation at a cell, that cell's capacitor is discharged. That action releases one side of the arrow flip flops surrounding that cell, causing those arrows to "point" to that cell with the discharged capacitor. The high outputs of the arrow flip flops then enter the neighbor cells, and begin discharging the capacitors there at rates determined by the voltages on the gates of  $Q_a$  and  $Q_b$ . When those capacitors are completely drained, the arrows surrounding those cells flip to point to the newly discharged capacitors, and the arrow outputs begin discharging capacitors in their neighbors. As the wavefront of activity propagates out, cells behind the frontier have completely discharged capacitors, cells ahead of the frontier have fully charged capacitors, and cells on the frontier have capacitors which are in the process of being discharged. The voltages on the gates of  $Q_a$  and  $Q_b$  are the crucial speed setting values. They are set by current mirror arrangements, as shown in Figure 11. There are three current mirror pads on the *PATHFINDER* chip, generating three control voltages for discharging the capacitors at the three rates required for the "easy", "hard", and "through" directions.

A feature included on the chip allows a small amount of local control over the cost function, to modulate the overall three costs described above. This consists of an additional pF or so of capacitance which can be switched on in parallel with the main capacitor in each cell. The time for propagating through a cell, and hence its propagation "costs", can be increased by connecting its extra capacitor before precharge and leaving it connected through propagation. The cost can be decreased by connecting the extra capacitor after precharge is over and disconnecting it again before propagation starts. These capacitor connections are switched on a cell by cell basis, controlled by a single bit in each cell. This makes it possible to increase costs near component pins, or to decrease costs near the board edges, etc., to reduce or increase the tendency for wires to end up in those areas. If circuitry had been included to discharge the extra capacitor when it was disconnected from the main one, additional levels of cost could be obtained by repeatedly connecting and disconnecting the extra capacitor between precharge and the start of propagation to remove more and more charge from the main capacitor, and thus reduce its discharge time. However, that extra feature was not included.

Figure 12 is a schematic of a two layer *PATHFINDER* processor, containing circuitry for the cells on both sides of the board as well as the arrow between them. The upper arrow and right hand arrow for each cell are arbitrarily assigned as belonging to that cell, while the lower and left hand arrows are considered to belong to the

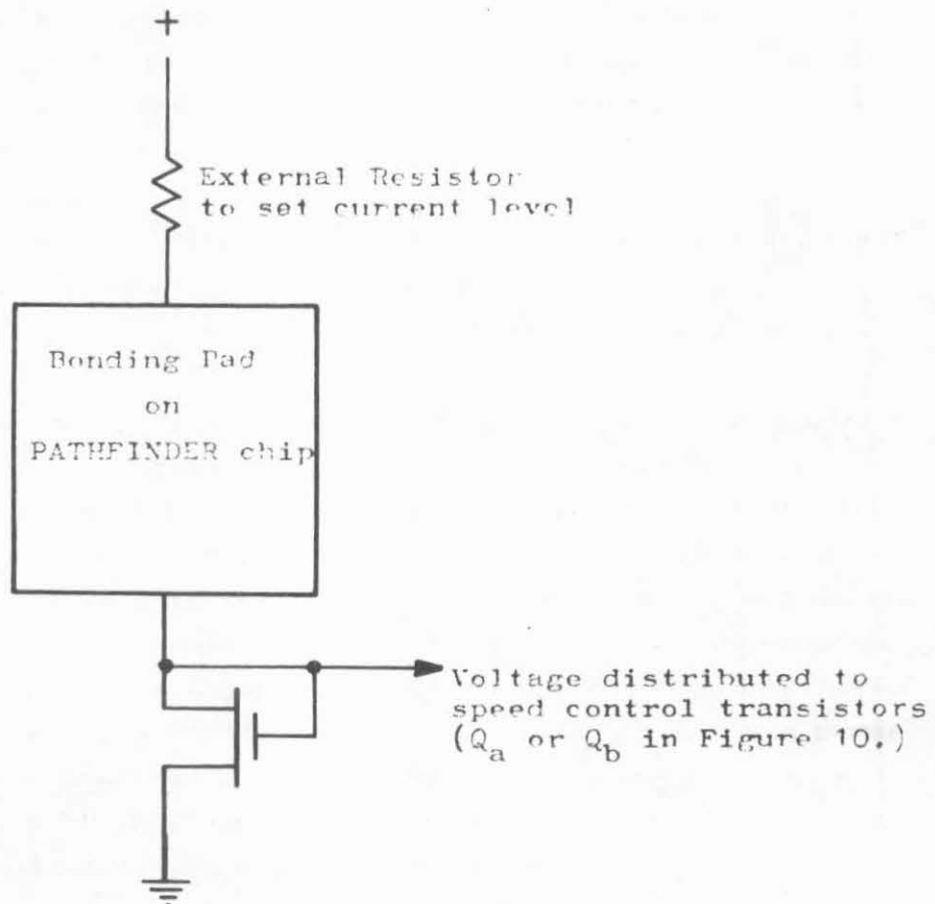


Figure 11. Example of a current mirror pad used on the PATHFINDER chip. There are three of these, one for each cost ("easy", "hard", and "through").

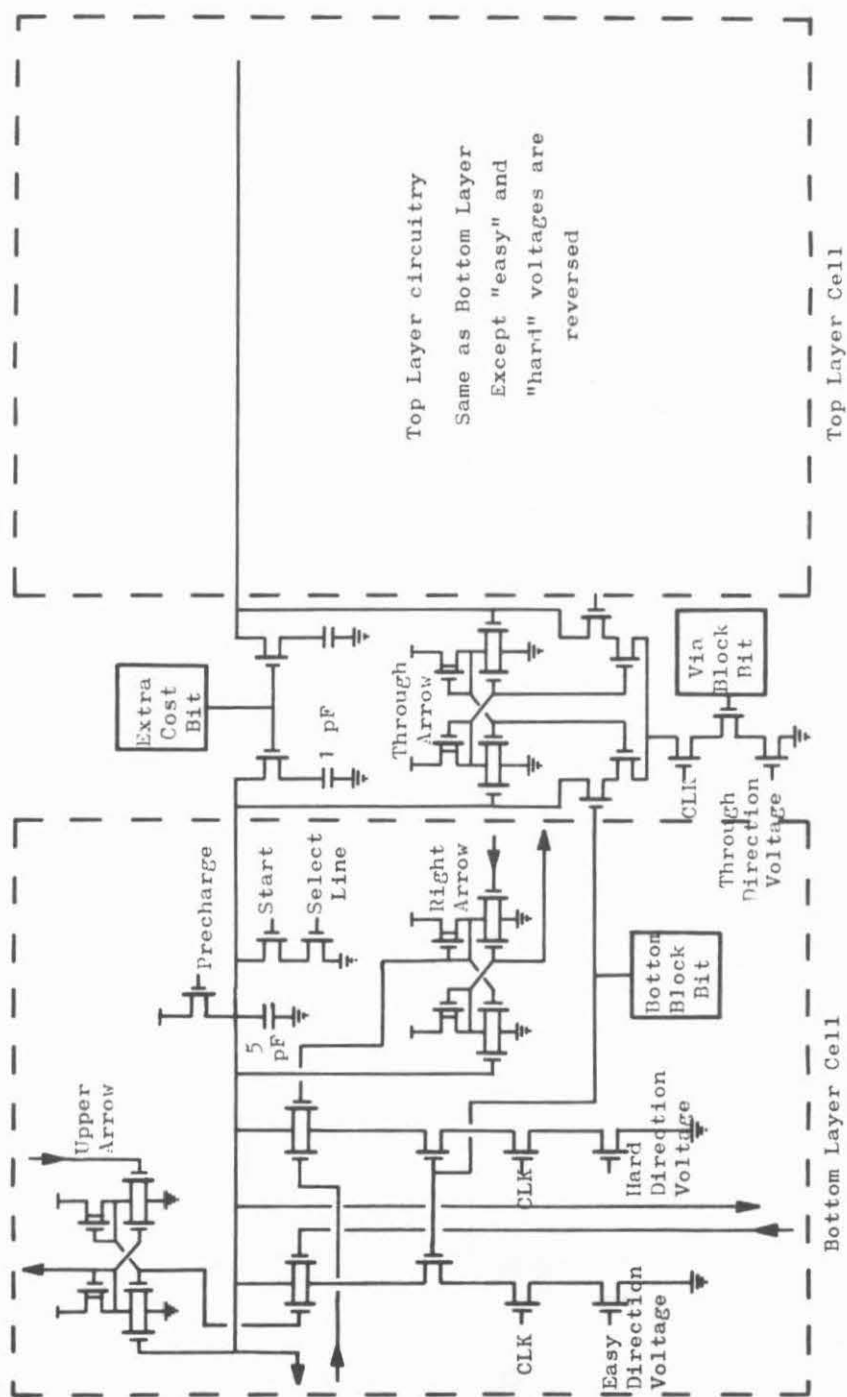


Figure 12. Schematic of PATHFINDER processor. Bit reading and writing mechanisms are not shown.

neighbor cells in those directions. The control storage bits are shown as boxes for simplicity. Actually the five arrow bits and the four control bits make up a nine bit word of what amounts to a standard static memory system, using the usual six transistor cell. Not shown are the two transistors which selectively link the flip flops to the word lines which run through all the bits, nor the select lines which control the gates of those transistors to do the addressing. Instead, the storage bits are shown located in reasonable places on the schematic to suggest their function in the circuit.

The circuit works just as described above for Figure 10, with the addition of the blocking controls and the switch to two layer operation. Having two layers merely means that three paths are present for discharging the capacitor, each controlled by a transistor whose gate voltage is set by one of the cost-setting current mirrors. The blocking control flip flops merely inhibit the appropriate discharge paths to prevent the discharge of the capacitor under the conditions which are to be blocked. The only remaining unexplained feature of the schematic is the signal labelled CLK. Remembering that this circuit was described as clock-less, one might wonder what the signal labelled CLK could be. In fact, it is a clock, but not in the usual sense.

The clock signal concerns a problem which has not been mentioned until now, having to do with chip boundaries in an array of cells composed of many chips. Because of the relatively large wiring capacitance associated with leaving one chip and entering the next, propagation from a cell on the periphery of one chip to its neighbor which happens to be on an adjacent chip will be much slower than propagation from cell to cell within the same chip. This can result in paths which have the problem shown in Figure 13, where the path takes every possible route to avoid crossing extra chip boundaries. The clock signal is an attempt to avoid this problem, by allowing the discharge process to occur only in short spurts. The clock is on for a few tens of nano-seconds, and then off for a hundred nano-seconds or so, giving the signals crossing from chip to chip time to settle. Letting the propagation process proceed only a little bit at a time like this slows down the system some, but should greatly alleviate the chip boundary problem.

Figure 14 shows a plot of the metal layer of the *PATHFINDER* chip. The chip contains a four by eight array of two layer processors. As with the *MAZER*, the large processor arrays of several hundred processors on a side which are needed for useful printed circuit board work are built up by assembling *PATHFINDER* chips themselves in an array. Forty-eight of the seventy pads are devoted to chip to chip

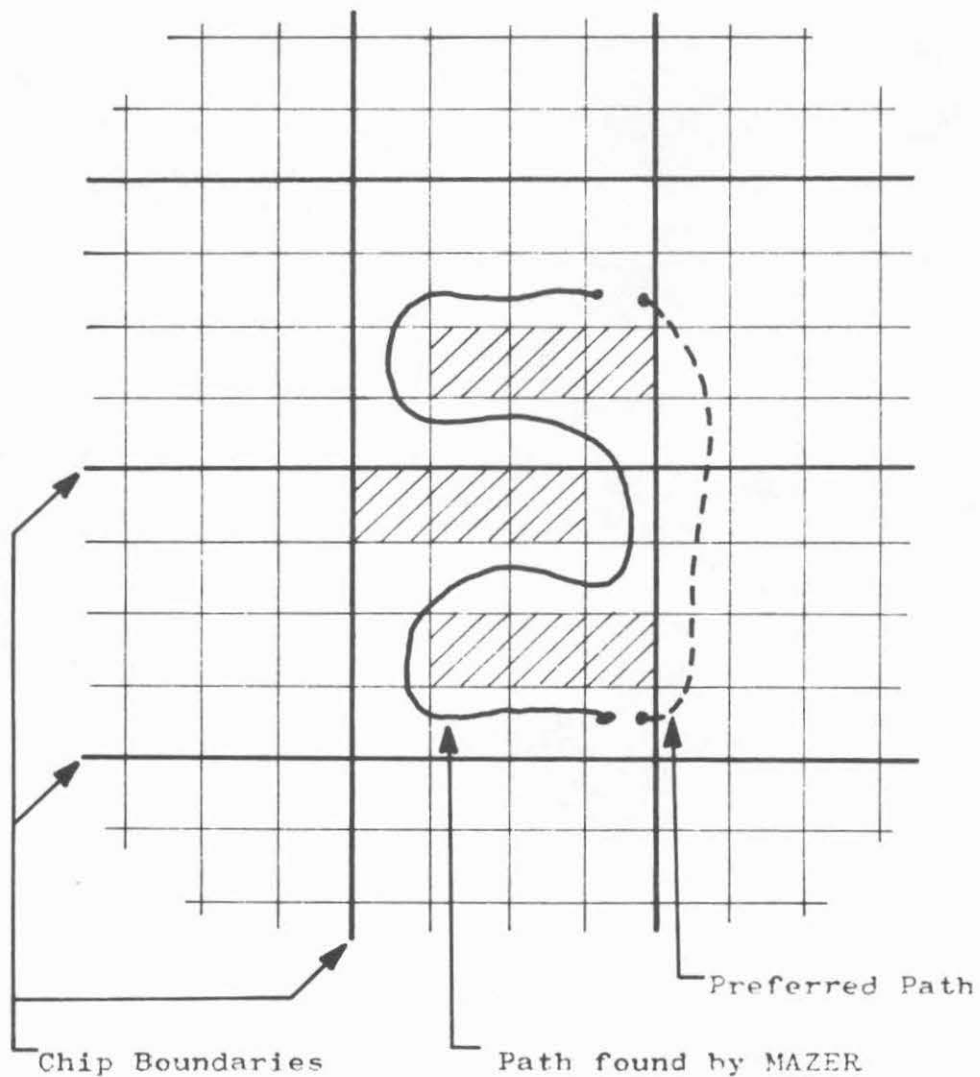


Figure 13. The problem posed by chip boundaries in large arrays

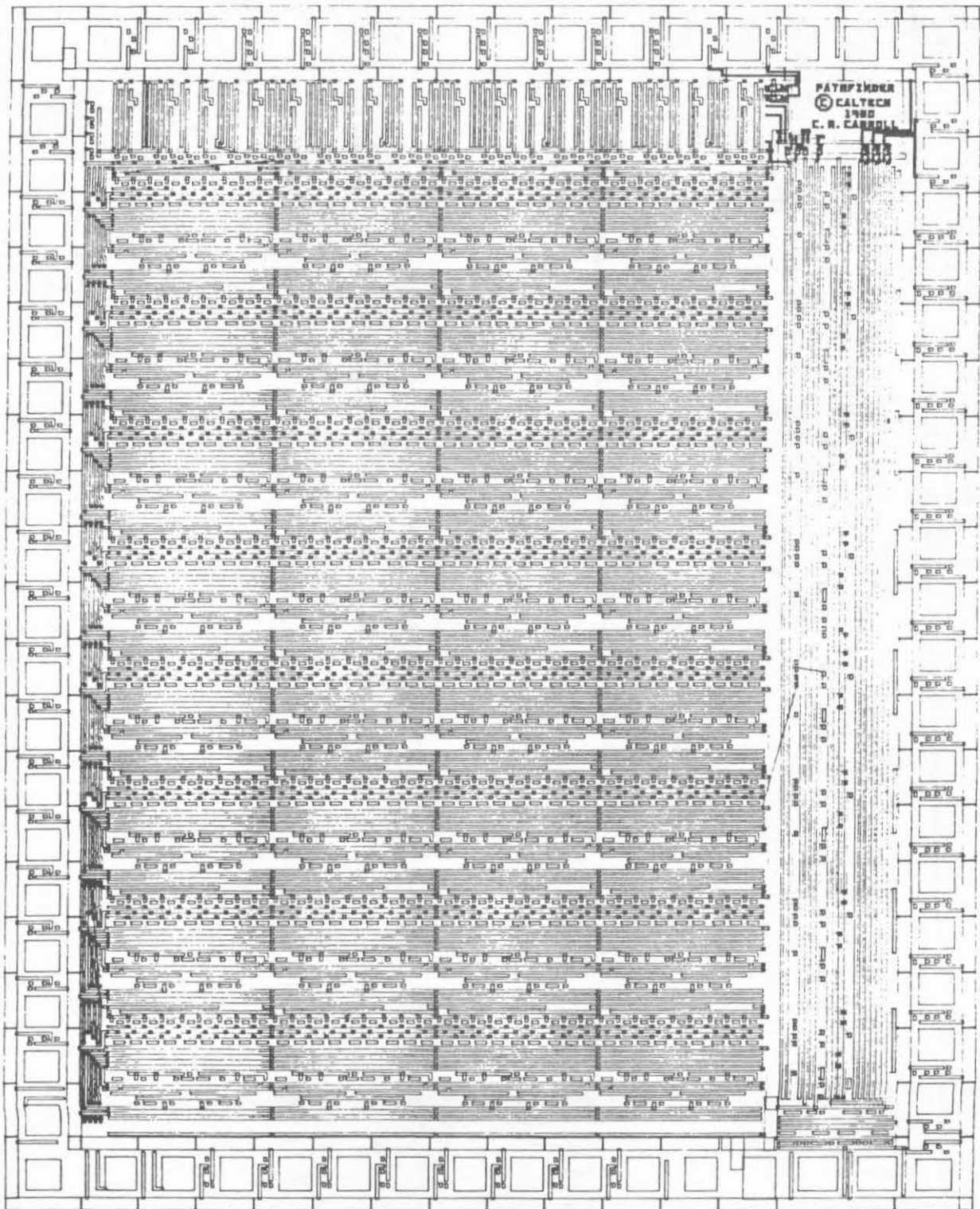


Figure 14. The PATHFINDER's metal layer

communication within the large array. The remaining pads consist of nine address pads, two power pads, two data I/O pads, and nine control pads, including the three current mirror cost-setting pads. Chip size is 3750 by 4875 microns.

### 3.2 PATHFINDER CHIP TEST AND CHARACTERIZATION

The *PATHFINDER* chip was included on the MPC380 run managed by Xerox PARC in the spring of 1980, and was also included on the M08B Mosis run managed by the Information Sciences Institute. From the two runs, I have received approximately twenty five copies of the chip. Only a few of the chips have been tested so far, however, because of difficulties in packaging the seventy pin circuit. Nevertheless, I do have one chip which is completely functional, and several others which work well enough to verify that the chip design is correct. Faults in the bad chips range from stuck bits to malfunctioning address decoders to complete failure, perhaps in the output buffers. Some of the chips have capacitors which fail to hold charge long enough to be useful. The capacitors in the one good chip, though, hold their charge long enough to keep the arrows "balanced" for about twenty seconds when carefully shielded from light. This is at least two orders of magnitude longer than required for successful path finding.

The chip has passed through several quantitative tests. Access time from chip enable to data out is around a microsecond, which is acceptable, though not noteworthy. An important item of interest is the operation of the cost-setting current mirrors. These perform very well. The range of control is excellent, with the normal external operating current between 100 and 1000 microamps. Characterizations of other quantitative aspects of the chip are not yet complete.

I have assembled a small microcomputer system to test the *PATHFINDER* chips and to operate them as a path finding system. With only one functional chip to use, no tests of the chip-to-chip communication strategy have yet been possible. However, I have written a true path finding program for the microprocessor which uses the one *PATHFINDER* chip to find paths through a four by eight grid. The program allows the user to set up any initial combination of blocked cells and blocked vias, start propagation from any cell in the array, and trace a path back to that starting cell from any of the other cells. The program displays the resulting path as well as the status of the control bits in the *PATHFINDER* chip on a terminal screen. In this implementation, three potentiometers, connected to the three current mirror pads on



the chip, are used to set the three global costs. The program can demonstrate the effect of changing costs on the path found between two points in the grid. For example, the user can cause the path to either skirt around barriers between the two endpoints, or to form vias and go over or under the barriers, depending on the settings of the three potentiometers. As more chips are tested and certified functional, this system can be expanded by connecting the good chips in an array to increase the size of the grid on which paths can be found.

### SUMMARY

This paper has detailed the design of hardware which implements the computationally expensive parts of the Lee-Moore path finding algorithm. The progression of designs, leading to the *PATHFINDER* chip, show that this is a natural application of array processing. Applied to the problem of two sided printed circuit board wire routing, the use of the chips described here can reduce computation time from several hours to around a minute. However, this circuit is innovative not only because of its array processing aspect, but also because of its unusual use of analog variables. These cost-setting control voltages are not there to simply make the circuit work, as is the substrate bias on the chip, for example. Instead, the values of the control voltages are important parameters to the computation performed in the processor array. Changing the values of the voltages changes the result of the computation. This application of analog processing in a digital system may be just the beginning of a new design discipline combining the advantages of the analog and digital design worlds.



### REFERENCES

1. E. Moore, "Shortest Path Through a Maze," Annals of the Computation Laboratory of Harvard University, Vol. 30. Cambridge, Mass.: Harvard University Press, 1959, pp. 285-292.
2. C. Lee, "An Algorithm for Path Connections and its Applications," IEEE Trans. Electronic Computers, Vol. EC-10, pp. 346-365, September, 1961.
3. S. Akers, "A Modification of Lee's Path Connection Algorithms," IEEE Trans. Electronic Computers, (Short Notes), Vol. EC-16, pp. 97-98, February, 1967.
4. Sutherland, Ivan, "A Better Mousetrap", Computer Science Department Display File #562, Caltech, March 8, 1977.
5. Slemaker, C., R. Mosteller, L. Leyking, A. Livitsanos, "A Programmable Printed-Wiring Router", Burroughs Corporation, Mission Viejo, California



# Special Purpose Hardware for Design Rule Checking

Larry Seiler

Massachusetts Institute of Technology

*Special purpose hardware can significantly increase the speed of integrated circuit design rule checking. The architecture described in this paper uses four custom chips to implement a raster scan DRC algorithm. It allows the use of  $45^\circ$  angles and can be programmed to check a wide variety of design rules involving an arbitrary number of layers. A shrink/expand operation allows the use of rasterization grids that are small relative to the minimum feature size. Using the Mead/Conway NMOS design rules<sup>8</sup> and assuming a grid size of  $1/2\lambda$ , or  $1/4$  the minimum transistor width, this hardware can completely check a  $3000\lambda \times 3000\lambda$  layout in under a minute, if the input data can be provided quickly enough.*

## 1. Introduction

One of the most computationally difficult aspects of integrated circuit design is the problem of checking for design rule violations. Design rules define the ways in which the features on the various layout masks may be positioned with respect to each other. In industrial applications, the problem is usually solved by running design rule checks as batch jobs on large machines. In university applications, the most common approach is to simplify the problem by using less complex design rules and disallowing nonorthogonal angles. Neither approach is completely satisfactory. What is needed is a method of design rule checking that allows the greater complexity of industrial design rules while retaining the speed and simplicity of the university design rule checkers.

Special purpose hardware is one way of satisfying these conflicting requirements. This hardware should be inexpensive enough that it can be included in individual color graphic designer workstations. It should be programmable for wide variety of design rules. It should be extensible to large numbers of layers and should be applicable to hierarchical design rule checking algorithms. Also, it should be able to handle  $45^\circ$  angles. Most industrial designs include  $45^\circ$  angles because they can result in a significant reduction in the area required by a layout. Allowing arbitrary angles provides only a small increase in packing density. The only significant use of angles which are not a multiple of  $45^\circ$  is in bipolar analog devices where circular wires are used to accurately control transistor ratios. It has been claimed that octagonal wires would be a sufficiently close approximation.<sup>5</sup>

This research was supported in part by United States Air Force Contract AFOSR-F49620-80-C-0073 and the Real Time Systems Group of the MIT Laboratory for Computer Science.

This paper is organized as follows. First, an overview is given of the algorithm used by the proposed design rule check hardware and the architecture that implements it. The next section describes algorithms that perform width checking and feature shrinking operations. A custom chip architecture that implements these algorithms is also described. Next, the boundary check operation is introduced for checking errors that depend on edge conditions. Another custom chip architecture implements this operation. A final section summarizes the work and suggests areas for further research.

## 2. Design Rule Check Hardware Description

The two main categories of design rule checking algorithms differ according to the types of objects they manipulate. Geometrical design rule checkers perform operations on geometrical objects such as rectangles, wires and polygons. Raster scan design rule checkers divide the layout into a grid of cells, each of which is empty or full on each layer. The simplest raster scan algorithms use a fixed size rectangular grid of cells, although variable sized cells and trapezoidal cells are also used.

The design rule check hardware described in this paper implements a fixed grid raster scan algorithm. This algorithm is especially well suited to hardware implementation because the data representation and the operations on the data are very simple. Also, the raster scan format includes local connectivity information, making expensive intersection tests unnecessary. This section starts by describing the basic algorithm, including its relation to hierarchical design rule checking. Next, the hardware architecture that implements it is described and the components of the architecture are discussed in greater detail. Finally, an estimate is developed for the speed performance of the architecture.

### 2.1 Top Level Algorithm

The basic structure of the algorithm is similar to one used in a software design rule checker written by Clark Baker.<sup>1</sup> Since IC layouts are usually described by a hierarchical structure of geometrical objects, the first step is to instantiate the hierarchy and create a raster image of the layout. Design rule checks are performed by moving a small rectangular window over each position in the rasterized layout, checking to see if the pattern in the window is valid at each position. For example, a 3x3 window could be used to find all places where a mask is only one unit wide. The pattern matching operations that are performed at each position are called local area design rule checks. The final step in the algorithm involves reporting the positions at which errors were found.

There are two main categories of local area design rule checks. The first is width and spacing checks, which are the most common operations. Shrink and expand operations on masks are also used. They are closely related to width checking. The other main category is general boundary checks. These window operations check the relationship between edges of features on two masks. For example, in the Mead/Conway design rules,<sup>8</sup> polysilicon and diffusion are only permitted to have coincident edges where they form a butting contact.

Local area design rule checks are not suitable for design rules that depend on mask features far away from the position being checked. Rules for pad size or spacing would require impossibly large windows. Rules that depend on the connectivity of the layout cannot in general be checked using windows, because there is no upper limit on the necessary window size. The algorithm will report all potential errors that depend on the connectivity or intended functionality of the layout so that a postprocessor can filter out the spurious errors from those that are genuine. Ideally, the postprocessor would include a node analysis step which would not only filter out potential connectivity errors but would also compare the intended circuit against the circuit that was actually implemented.<sup>7</sup>

What is commonly referred to as hierarchical design rule checking is not so much a method of performing design rule checks as it is a method of reducing the work required by a nonhierarchical algorithm. It is important to consider whether the basic algorithm defined above can be used in a hierarchical design rule check algorithm. One such algorithm operates by removing duplicate geometry from the layout.<sup>11</sup> This would not significantly decrease the running time of any raster scan algorithm because the number of geometrical objects in the layout is reduced rather than the area that the layout occupies. However, it is possible to design a hierarchical algorithm that does reduce area. An example is an algorithm that checks a primitive cell in the usual way and checks a nonprimitive cell by checking its subcells and then checking the areas where its subcells overlap. In any case, hierarchical design rule checking can be done using raster scan algorithms as well as geometric algorithms.

## 2.2 Top Level Architecture

The design rule check architecture described below implements the inner loop of the above algorithm. Custom chips implement primitive operations involving rasterization and design rule checking. Standard MSI and memory parts complete the system. The resulting hardware will be able to fit on a single PC board and will be inexpensive enough to include as a part of individual color graphic designer workstations. First, the basic architecture is described along with the operations it performs. Next, the interface between the portions of the algorithm implemented in hardware and software is discussed. The extensibility of this architecture to larger layouts and more complex design rules is also considered.

Performing a design rule check with the proposed hardware requires three distinct operations, as illustrated in figure 1. These three operations are performed for each scan line. First the design must be converted into a raster image. This is done by feeding mask features that intersect the current scan line into custom chips in the rasterization unit. The resulting line of rasterized mask data is given to a unit which performs local area design rule checks. This unit performs boolean operations on the rasterized input masks, buffers previous raster lines of the derived masks, and feeds them into two kinds of custom chips that perform primitive DRC operations. The output from the local area DRC unit consists of parallel streams of error bits, where a one indicates an error at that position in the layout. The error reporting hardware converts this into a sequence of error coordinates which are read by the controlling processor. Assuming that the incidence of errors is low, this will result in a significant compression of the error information.

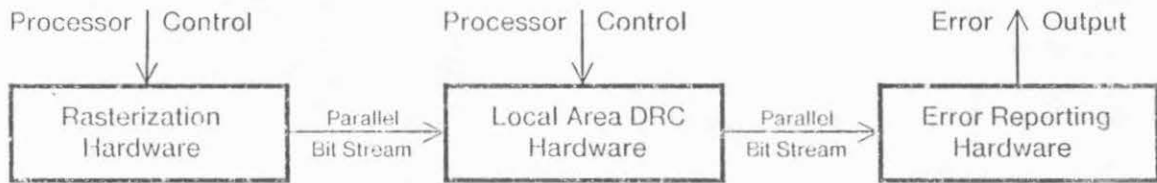


Figure 1: Design Rule Check Hardware System

The hardware architecture in figure 1 cannot stand alone, but must be a part of a design rule checking system. This system contains a controlling processor which includes software to program the local area design rule check unit, convert the layout into the proper format for the rasterization unit, and convert the error reports into human readable output. The design rule check hardware simply speeds up the inner loop of a software algorithm. The basis for choosing these interface points between hardware and software involves the volume of data which is manipulated and the complexity of operations on that data. Software routines can do general data manipulation operations but are relatively slow at processing large amounts of data. On the other hand, data can be manipulated very quickly using special purpose hardware, but the hardware cost increases rapidly as the complexity of the operations increases. The input and output of the local area DRC unit is rasterized data streams, which are easy to process using hardware but would be very time consuming to process using software. The data manipulations required to instantiate the layout preparatory to rasterization and to process the error positions for user output are very general and involve much smaller quantities of information. With the hardware software interface shown above, the hardware implements simple data manipulations on large volumes of data and the software implements general operations on smaller volumes of data.

The above architecture can handle a wide range of layout sizes and design rule complexities. The parameters that limit the size and complexity that can be handled are the number of rasterization chips, the number of parallel mask data lines, the size of the line buffers, and the number of custom chips in the local area design rule check unit. None of these factors place a serious limitation on the usefulness of the hardware. A layout which is too wide for the line buffers can be split into parallel strips that can be checked separately. Adjacent strips must overlap by an amount equal to the largest design rule size. Statistical studies permit good estimates to be made as to the number of mask features that will intersect a scan line.<sup>4</sup> If there are too many features for the rasterization chips to handle, a smaller line size can be chosen. It is not necessary to check all of the design rules during one pass through the layout, so the local area design rule check hardware need only include enough custom chips to handle each design rule individually. Finally, only those masks that are actually being used need to be sent to the local area check hardware during a given pass through the design, so the number of parallel mask data lines does not need to be as large as the total number of masks.

### 2.3 Detailed Architecture

This section describes the rasterization, error reporting, and local area design rule check units in greater detail. The architecture of the local area design rule check unit is described in the most detail, with sample bus sizes and chip counts.

The rasterization unit outputs parallel streams of rasterized mask data to the local area DRC unit. Its input consists of the set of intervals on the current scan line that are covered on each mask. For orthogonal rectangles, this requires that the controlling processor add an interval to the set when the current scan line reaches the start of the box and remove it when it reaches the end. Trapezoids with two horizontal sides and two sides at  $45^\circ$  angles can be rasterized by adding an interval to the set when the current scan line reaches the start of the trapezoid, and then incrementing or decrementing the ends of the interval before each successive scan line until the end of the trapezoid is reached. Polygons and wires must be decomposed into trapezoids. The interval rasterization operation is performed by a custom chip, similar to a chip designed by Bart Locanthi.<sup>6</sup>

The error report unit is the interface between the local area DRC unit and the controlling processor. It converts the parallel streams of error bits into a list of positions where errors were discovered. The type of error at each position is also reported. Most of the error bits will be zero, so this will significantly reduce the amount of information passed to the controlling processor. Typically, the error positions will be saved on a disk file for further processing after the design rule check is complete.

The local area design rule check unit accepts parallel streams of mask data bits and produces parallel streams of error bits. It performs primitive DRC functions such as width tests and boundary checks. It also implements mask shrink and expand operations and boolean operations such as mask intersections, unions, negations, and differences. The boolean operations are used to create derived masks such as transistor gate area, which is defined to be the intersection of the polysilicon and diffusion masks. Since the width tests, boundary checks, and shrink/expand operations require looking at the masks through windows of size up to  $4 \times 4$ , buffers are used to save up to three previous lines of each mask. Figure 2 illustrates the architecture of the local area design rule check unit with sample bus sizes and numbers of chips.

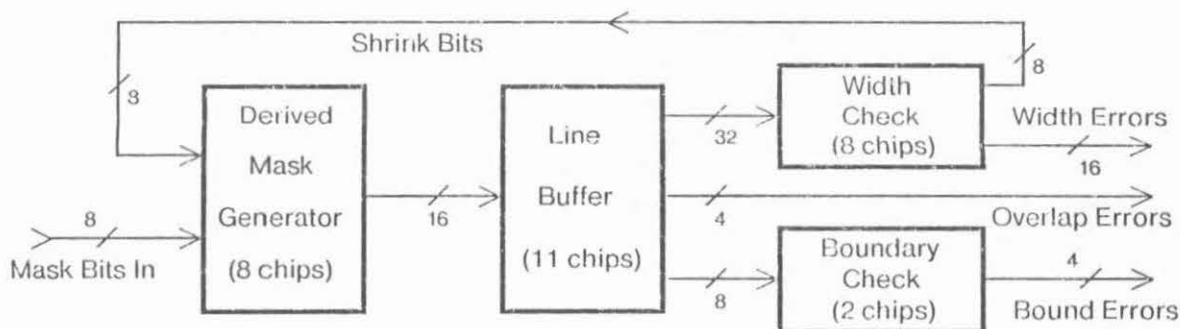


Figure 2: Local Area DRC Hardware



The mask bits input to the the local area DRC unit are fed into the derived mask generator, which uses a custom chip to perform boolean operations on them. The derived masks are fed into the line buffer, which uses standard memory parts to buffer preceding lines. Each custom chip in the width check unit receives a mask from the derived mask generator along with the corresponding bits in the preceding three lines saved in the line buffer. The width check chips each output two width error lines for the input mask and also output the result of a shrink operation on the mask. This operation can also be used for mask expansion. The shrink output is fed back around to the derived mask generator, allowing multiple shrink operations to be done. The boundary check chips each require bits from the current and preceding lines of two derived masks and produce two error outputs. The final group of error lines are output directly from the derived mask generator. These are useful for situations such as overlap tests, where errors are found by subtracting one mask from another. The algorithms and architectures used for the width check and boundary check chips are described in sections 3 and 4, respectively.

## 2.4 Timing Estimate

At this point, we have enough information to estimate the speed of the design rule check hardware. The basic data rate is determined by the rate at which the rasterizer sequences through the positions on a scan line. Assuming that data is buffered at appropriate places, the most complex operation that must be done during a single cycle is a memory read and write in the line buffers. It is reasonable to assume that this can be done in 200ns. Using a grid size of  $1/2\lambda$ , and given the hardware configuration in figure 2, the Mead/Conway design rules can be checked in no more than five passes through the design rule checker.<sup>10</sup> Assume that the chip being checked is  $3000\lambda \times 3000\lambda$ , which is 295 mils on a side if  $\lambda$  equals 2.5 microns. Further assume that there is a 50% overhead resulting from overlapping strips and delays between scan lines. The equation below gives the time needed for a complete design rule check.

$$2.1 \quad (3000\lambda)^2 \cdot 200\text{ns} / (1/2\lambda)^2 \cdot 5 \cdot 150\% = 54 \text{ seconds}$$

Of course, the controlling processor will not necessarily be able to provide data to the rasterization unit that quickly. Clark Baker's raster scan DRC program takes 49 seconds simply to read the instantiated rectangle file for a chip that size, which contains about 100,000 rectangles.<sup>2</sup> This would have to be done once for each pass through the design rule checker. Experience indicates that it would be much faster to instantiate the chip on the fly from a hierarchical description, rather than read it from a large disk file. Further research must be done to find ways of quickly getting data into the rasterization unit. If software instantiation algorithms are not fast enough, special purpose hardware could be designed to speed that up as well.



### 3. Width Checking and Feature Shrinking

The most frequent operation in integrated circuit design rules is minimum width checking. Polysilicon, diffusion, metal and contact cut masks all have their own minimum feature size. Spacing checks are simply width checks performed on the complement of a mask, so spacings between features on the same or different masks can be checked in the same way that widths are checked.

This section starts by describing how to check for width errors in mask features which are orthogonal and are small in comparison to the grid size. Then the method is expanded to handle edges at 45° angles. A feature shrinking operation is introduced to allow checking of greater widths. This operation can also be used to expand masks. Next, a custom chip is described that implements these operations. Finally, a notation is developed for specifying width checks and shrink or expand operations.

#### 3.1 Orthogonal Width Checking

Width checks require looking at features in a window which is one greater in size than the width that is being checked. So, a 3x3 window is needed to verify that a mask is at least two units wide and a 4x4 window is needed to check for width three. Figure 3 illustrates the set of patterns that implement these checks. A one indicates a position where the mask is required to be present, a zero indicates a position where the mask is required to not be present, and a dash represents a don't-care position. The mask passes the width 2 test if it matches one of the four orthogonal rotations of one of the patterns in figure 3a. The mask passes the width 3 test if it also matches one of the four orthogonal rotations of one of the patterns in figure 3b.



Figure 3a: Valid 3x3 Windows

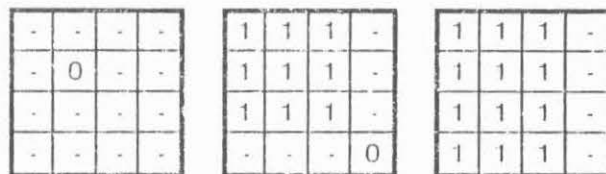


Figure 3b: Valid 4x4 Windows

The patterns in figure 3a are correct because a mask fails the width 2 check if and only if it contains a feature of width 1. The patterns in figure 3a check whether the center cell is part of a feature of width 1. If a mask matches the first pattern in figure 3a, then the mask has width zero at this point. If it matches the second pattern, it is part of the corner of an area which is at least 2 units wide. If it matches the third pattern, then it is part of the center or edge of an area which is at least 2 units wide. Therefore, if it does not match any of these patterns, it must be part of a feature which is only one unit wide.

The justification for the patterns in figure 3b is similar. They check whether the center 2x2 box is part of a feature which is greater or less than 2 units wide. If there is a zero in one of the four center cells, then the mask is less than two units wide at that position. If the mask matches the second or third patterns, then the present position is the corner, edge, or interior of an area which is at least three units wide. If none of these patterns are matched,

then there must be a feature of width 2 at this position. If a pattern in figure 3a and a pattern in figure 3b are matched, then the mask must be either less than 1 or greater than 2 units wide. This is the test that is needed for the width 3 check.

Figure 4 gives examples of checking for width 2 and width 3. The edges and corners of the rasterized rectangles must fall exactly on the rasterization grid for the width check to work correctly. Rounding is not permitted. Each example contains an error which is marked by an X. The 3x3 and 4x4 windows show the mask pattern around the error positions. It should be noted that if the zero were not present in the corner patterns in figure 3, the errors below would not be detected.

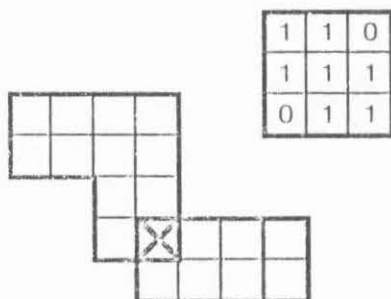


Figure 4a: Width 2 Example

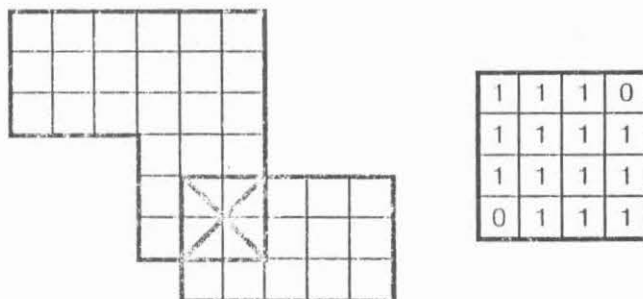


Figure 4b: Width 3 Example

There is some question as to whether the pattern in figure 4b is actually an error. The angular distance across the stricture is  $2 \cdot (2)^{1/2}$ , which will be referred to as 2 diagonal units or 2 diags. This distance is approximately equal to 2.83 orthogonal units. This is only 6% less than the required width of 3 units, and in some fabrication processes it could be considered sufficiently close as to not be an error. To retain the greatest degree of generality it must be possible to select whether or not this case represents a width 3 error.

### 3.2 Angled Width Checking

Figure 4 gave an example of measuring the width of a feature along a 45° angle rather than orthogonally. Now we will consider how layouts that include 45° angles may be checked for widths 2 and 3. To be checked correctly, all edges must fall on the grid illustrated in figure 5a. Figures 5b and 5c show two more cases that must be allowed in order to do width 2 and 3 checks on a mask with edges at 45° angles.

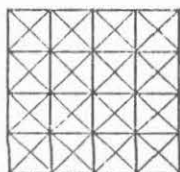


Figure 5a: 45 Degree Grid

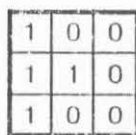


Figure 5b: 3x3 Angled Corner

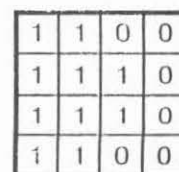


Figure 5c: 4x4 Angled Corner

The rules for rasterizing 45° edges are very simple. When a width check is going to be performed, all partially covered raster cells are filled in. When a spacing check is going to be performed, all partially covered cells are marked empty. Figure 6 shows a width check

being performed on a wire which contains a  $45^\circ$  bend. When the wire is rasterized, the  $45^\circ$  edges become stairsteps which touch actual edges of the wire at each step. The result is that the rasterized wire is at every point greater than or equal to the correct width, allowing a width check to be done without reporting spurious errors. Since the rasterized wire has the correct width once at each staircase, any genuine errors will be discovered. The different rasterization rule for spacing checks insures that all spacings are greater than or equal to the correct value. The X in figure 6 marks the position which is illustrated in the window at the right. If the angled corners in figure 5 were not added to the set of valid patterns, this would be reported as an error.

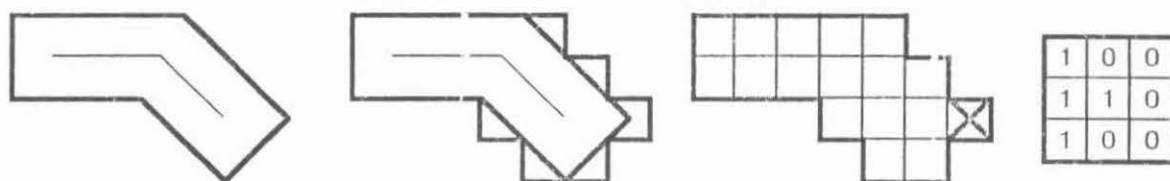


Figure 6: Angled Width 2 Example

Figure 7 gives an example of checking an angled wire of width 3. The width of the angled portion of the wire is 2 diags, or 2.83 units. The upper left X and the upper left window illustrate that this will be detected as a width 3 error unless the pattern in figure 4b has been specified as valid. The lower right X and the lower right window show an example of the 4x4 angled corner from figure 5c.

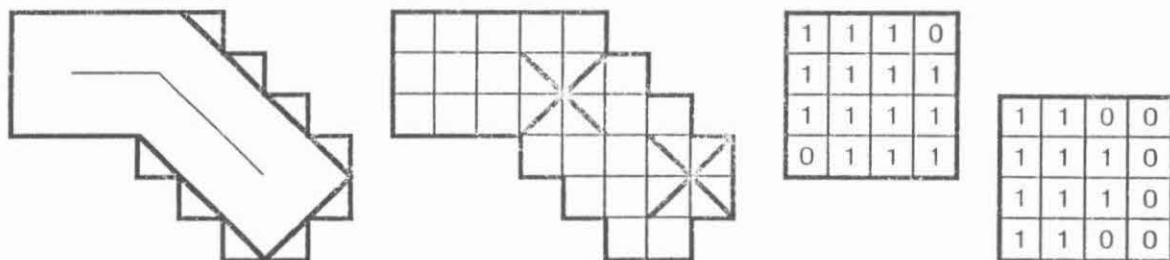


Figure 7: Angled Width 3 Example

Allowing the angled corner patterns in figure 5 causes a problem which is illustrated in figure 8. These patterns cannot be distinguished from angled corners. As a result, these errors are not detected if angled corners are allowed. A strong case can be made that these errors are insignificant. However, it is possible to detect these errors and still allow angled corners by doing two width checks. The first check allows angled corners and the second does not. If an error occurs in the second width check that did not occur in the first, and that position is not the corner or edge of an angled box or wire, then it must be one of the errors in figure 8.

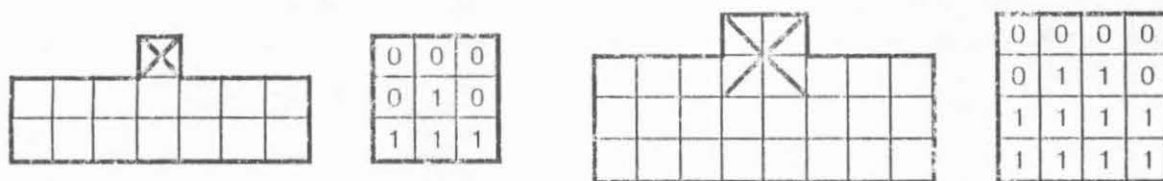


Figure 8: Undetected Errors

The inclusion of  $45^\circ$  edges can result in some undesirable degenerate cases. Figure 9 shows what happens when  $45^\circ$  edges are coincident or within  $1/2$  diag of each other. Figure 9a illustrates a zero width angled wire which is rasterized for a width check and two angled wires with coincident edges which are rasterized for a spacing check. In the first case, a zero width line causes raster cells to be filled in. In the second case, unwanted holes appear along touching edges. The remedy for these problems is to require the rasterization algorithm to detect coincident  $45^\circ$  edges and either ignore the feature or fill in the raster cell at that point, depending on which of the cases in figure 9a has been detected.



Figure 9a: Coincident Edge Degenerate Cases



Figure 9b: Near Edge Degenerate Cases

Figure 9b shows what happens when  $45^\circ$  edges fall  $1/2$  diag apart. In the first case, a wire which is only  $1/2$  diag wide disappears completely upon a spacing check rasterization. If two angled wires have edges that fall  $1/2$  diag apart, the gap disappears during a width check rasterization. These problems can be solved by doing both a width 2 check and a spacing 2 check on each mask that may have  $45^\circ$  edges. Checking the structure on the left, for example, will cause a width 2 error to be reported, even though the spacing error that might exist will not be reported. The only complication comes with a mask such as depletion mode implant, which does not have any minimum width or spacing rules. One way to solve this problem is to impose width 2 and spacing 2 rules on this layer and accept the spurious errors that might result. Most of the spurious errors may be filtered out by having the rasterization algorithm report points where  $45^\circ$  edges are  $1/2$  diag apart. Any legitimate error will be recognized as such by both checks. It should be noted that this is only necessary if  $45^\circ$  edges are allowed in the implant mask. If  $45^\circ$  edges are used for interconnect only and not for transistor or implant areas, then this case will not arise.

Figure 9 also illustrates how the rasterization algorithm deals with acute angles. An inside acute angle will always be detected as a width error and an outside acute angle (an acute angle on the complement of the mask) will always be detected as a spacing error. It would be possible to recognize the occurrence of acute angles and not flag them as errors, but there is little point in doing this. It is not possible to accurately pattern an acute angle onto silicon, so it is not very useful to include one in a layout.

### 3.3 Feature Shrinking

The algorithms developed above allow mask features to be checked for width 1 errors or width 2 errors using 3x3 and 4x4 windows, respectively. Larger windows could be used to check for greater widths. However, the complexity of the width checks goes up rapidly as the size of the window increases. Also, there is no fixed limit to the widths that will need to be checked. What is needed is a way of making the width check function modular. This can be done by introducing feature shrinking.

The goal of the feature shrink operation is to reduce the size of the features in a mask so that the width 2 or width 3 check can be used. Feature shrinking is done by passing a 3x3 window over the selected mask and producing an output which is one or zero depending whether the mask matches a specified pattern, as for the width 2 check. The difference is that the output is used as another rasterized mask rather than as an error indication. Figure 10 illustrates the four patterns that are used. The orthogonal shrink and angled shrink patterns are discussed below. The shrink/expand and null shrink patterns are discussed at the end of this section. Note that any of the four rotations of the shrink/expand pattern constitute a match.

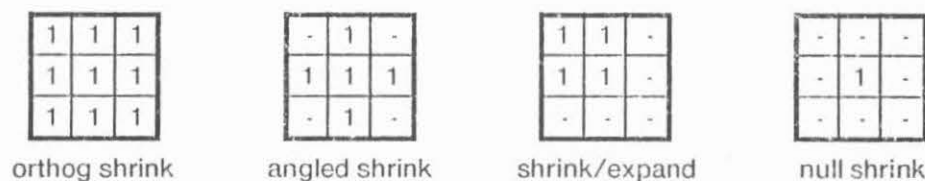


Figure 10: Patterns for Feature Shrink

If the orthogonal shrink pattern is applied to a mask, the resulting output will be a one whenever the center of the pattern is in the inside of a feature of width 3 or more. If it is on the edge or corner of a mask feature or is outside, the output will be zero. The result of this is a mask which is the same as the input mask except that all of its horizontal and vertical edges have receded by one unit and all of its angled edges have receded by one diag. Any parts of the mask that are less than three units wide will disappear completely. The angled shrink is almost the same, except that it causes 45° edges to recede by only 1/2 diag. This is illustrated in figure 11. The orthogonal portion of the wire starts out 4 units wide and the 45° portion starts out 3 diags wide. After either shrink, the orthogonal portion of the wire is 2 units wide. After the orthogonal shrink the angled portion is only 1 diag wide, which is a width 2 error, but after the angled shrink it is 2 diags wide. The name of the angled shrink pattern refers to the fact that it does not reduce angular widths as much as the orthogonal shrink.



Figure 11: Feature Shrink Example

The local area DRC architecture in figure 2 shows that the shrink outputs from the width check chips are fed back into the derived mask generator for combination with other masks. Therefore, a single mask may be shrunk or expanded as many times as there are width check chips, reducing its width by 2 units each time. This makes it possible to check arbitrarily large widths using only the width 2 and width 3 checks. Since mask features of width less than 3 disappear during the shrink operation, a width 3 check must be performed each time a mask is shrunk. The same patterns that are used to shrink masks can also be used to expand masks, since the expansion of a mask is simply the complement of the shrink of the complement of the mask.

The patterns in figure 12 describe double shrink operations. Shrinking a mask twice using the orthogonal or angled shrink patterns is equivalent to shrinking it once using one of the four double shrink patterns. The two center patterns demonstrate that the order in which a sequence of orthogonal and angled shrinks are done is unimportant because the two shrink operations are commutative. The patterns in figure 12 can also be interpreted as the result of expanding a mask which had a one in the center of the window. Orthogonal expansions produce squares, angled expansions produce angled squares, and a combination of the two produces octagons. This shows that the degree of orthogonal and angular shrinkage or expansion can be selected independently.

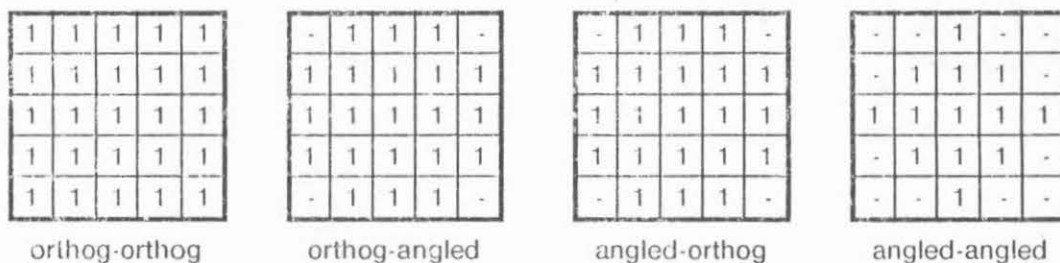


Figure 12: Double Shrink/Expand Patterns

The shrink/expand pattern in figure 10 outputs a one if the cell in the center of the window is part of a feature of size 2x2 or greater. This does not result in any shrinkage of the mask. Instead, it removes features of width 1, as if the mask had been shrunk by half a unit in each dimension and then expanded back again. Figure 13 illustrates using this operation to find the active area of a transistor using the Mead/Conway design rules.<sup>8</sup> The layout on the left depicts a depletion mode pullup transistor combined with a butting contact.



Only the polysilicon and diffusion masks are shown. The center figure illustrates the intersection of these two masks, which includes the unwanted butting contact strip. The shrink/expand operation results in the mask on the right, which is the real transistor gate area. An orthogonal shrink followed by an orthogonal expand can be used to remove areas of width 1 or 2. An angled shrink followed by an angled expand removes features of width 1 or 2 and also rounds off orthogonal corners.

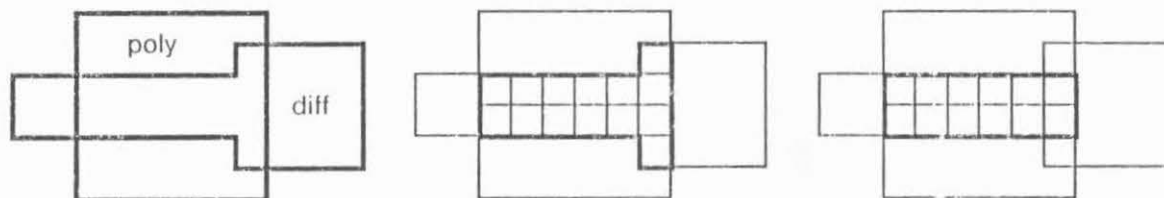


Figure 13: Shrink/Expand Example

Overlap tests can be done by subtracting a shrunk mask from an expanded mask, using the feedback lines into the derived mask generator. However, a shrunk or expanded mask cannot be combined with an unchanged mask. The reason is that the shrink operation displaces the mask that it operates on, since the current mask position is in the corner of the window and the shrink pattern works on the center. The bit entering the derived mask generator from a feedback line is not at the same position on the layout as a bit coming directly from the rasterizer. To combine an unchanged mask with a mask that has been shrunk or expanded, it is necessary to displace it by the same amount as a shrink or expand operation. The null shrink pattern in figure 10 does this. The only change it causes in the input mask is to displace it by the same amount as a shrink or expand. In section 3.4, we will see that there is another use for the null shrink pattern.

### 3.4 Width Check/Shrink Chip

The previous sections have defined the functionality that is necessary for the width check/shrink chip. Figure 14 gives a structure that implements this functionality. All of the window patterns are implemented in a clocked PLA which has 44 AND terms. Four successive rows of mask data are input to the chip and the previous three values from each row are saved so that a 4x4 window is input to the PLA. Five control lines are also input to the PLA. Their exact functionality is described below. The output lines WIDTH 1 ERROR and WIDTH 2 ERROR indicate when one or two unit wide mask features are found. SHRINK 1 OUT and SHRINK 2 OUT are the result of applying the shrink operation selected by the control lines. The first applies it to a 3x3 window which uses rows 1-3 and the second applies it to a 3x3 window which uses rows 2-4. This allows the width check chip to output two rows of shrunk mask in parallel. SHRINK 1 OUT can be used as a feedback line to the derived mask generator. A use for SHRINK 2 OUT will be described at the end of this section.

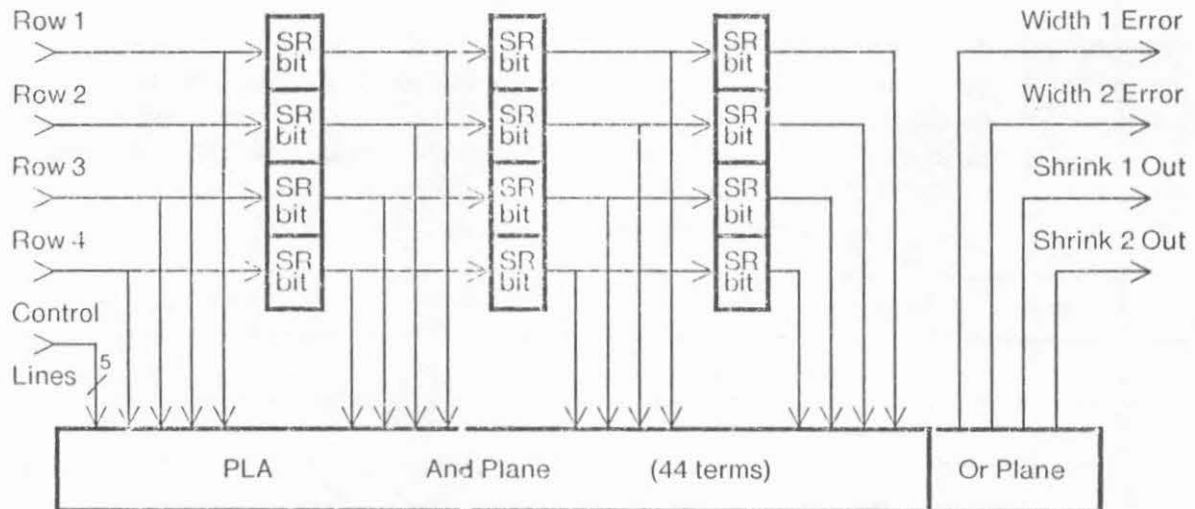


Figure 14: Structure of Width Check/Shrink Chip

Three of the five control lines specify the kind of width check that is performed. These inputs are called WIDTH SELECT, ANGLE OK, and CORNER OK. The WIDTH SELECT line chooses between width 2 and width 3 checking. The WIDTH 2 ERROR output is only nonzero when width 3 checking is selected. This way, the two width error outputs can be OR'ed together externally to produce a single width error signal if the separate information is not needed. The ANGLE OK line controls whether the angled width patterns in figure 4 are treated as width errors. The CORNER OK line causes the angled corners in figure 5 to be accepted. The remaining two control lines specify the pattern that is used to produce the shrink outputs. These inputs are called SHRINK SELECT 1 and SHRINK SELECT 2. Table 1 shows which shrink pattern is used for each combination of the select lines.

Select 1	Select 2	Shrink Type
true	true	orthog shrink
true	false	angled shrink
false	true	shrink/expand
false	false	null shrink

Table 1: Pattern Selection for Feature Shrink

For design rules that contain many large widths and spacings, it would be good to be able to do a double shrink without using feedback lines. Figure 15 shows how several width check/shrink chips could be combined to do double or even triple shrinks at once, reducing the width of mask features by 4 or 6 units at a time. Note that the double shrink requires 6 input rows instead of 4 and the triple shrink requires 8 input rows. Also, a double or triple shrink configuration can be set to do a single or double shrink by specifying a null shrink in the first column of width check chips. All of the control lines in a single column should be tied together so that the same type of shrink is done by all chips in the same column.



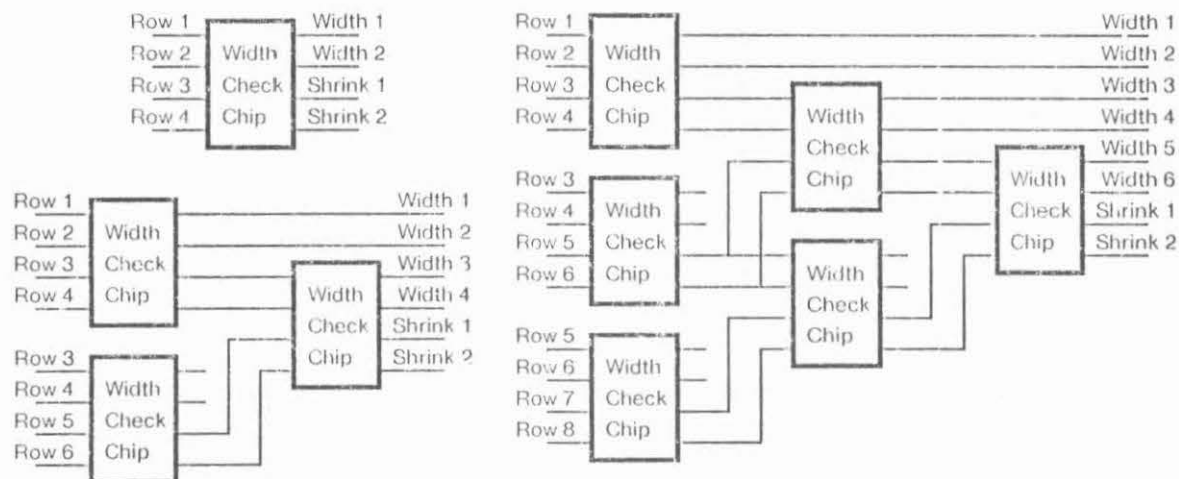


Figure 15: Single, Double, and Triple Width Check Configurations

### 3.5 Width Check Specification

Now that a custom chip has been defined that can do small width checks and several kinds of feature shrinking, it is necessary to define how larger checks are done. This section shows how to do arbitrarily large width check and feature shrink operations using multiple width check/shrink chips and then defines a notation for specifying width checks and shrink operations.

When a width check is done involving one or more width check/shrink chips, all but the last chip should select width 3 checking. The value of the CORNER OK line should be the same for all of them. ANGLE OK should be false for all except possibly the last one. The number of chips which should have ORTHOG low to select angled shrinking and the state of the ANGLE OK and SELECT lines for the last width check chip depend on the specific width check that is being performed. Table 2 gives the values to use for doing width checks on features up to size 6. A width check is determined by the required orthogonal and angular widths. For example, the sixth line of the table tells how to check for features with a minimum orthogonal width of 4 units and a minimum 45° width of 3.0 diags, or 4.24 units. The table is sufficiently large that the pattern should be clear. The final width check is width 2 if the orthogonal width to be checked is even and width 3 otherwise. Specifying ANGLE OK for the last chip reduces the required diagonal width by 1/2 diag. Changing an orthogonal shrink into an angled shrink reduces the required angled width by 1 diag. This is because the orthogonal shrink causes angled edges to recede by 1 diag while the angled shrink causes them to recede by only 1/2 diag.

orthogonal width	angular width diags	units	orthogonal shrink	angled shrink	final select	final angle ok
2	1.5	2.12	0	0	width 2	no
2	1.0	1.41	0	0	width 2	yes
3	2.5	3.54	0	0	width 3	no
3	2.0	2.83	0	0	width 3	yes
4	3.5	4.95	1	0	width 2	no
4	3.0	4.24	1	0	width 2	yes
4	2.5	3.54	0	1	width 2	no
5	4.5	6.36	1	0	width 3	no
5	4.0	5.66	1	0	width 3	yes
5	3.5	4.95	0	1	width 3	no
6	5.5	7.78	2	0	width 2	no
6	5.0	7.07	2	0	width 2	yes
6	4.5	6.36	1	1	width 2	no
6	4.0	5.66	1	1	width 2	yes

Table 2: Control Line Values for Width Checking

Now it is possible to define a notation for specifying width checks and shrink operations.  $W2(\text{mask})$  represents a width 2 check, that is, errors are reported for features of size 1. Subscripts are used to indicate the required diagonal width. For example,  $W4_{3.5}(\text{mask})$  indicates a check for features with an orthogonal width of 4 units or an angled width of 3.5 diags. The fifth line of table 2 describes how this width check could be achieved. Shrink and expand operations are expressed similarly.  $S1(\text{mask})$  describes a shrink operation which causes each edge to recede by one unit.  $S3_{2.0}(\text{mask})$  denotes a shrink of 3 orthogonal units and 2 diagonal units. This requires one orthogonal shrink and two angled shrink operations.  $E1_{0.5}(\text{mask})$  specifies an angled mask expansion and is equivalent to  $\neg S1_{0.5}(\neg \text{mask})$ .  $N1(\text{mask})$  indicates a single null shrink, which displaces the mask by the same amount as a shrink operation but otherwise leaves it unchanged.  $SE(\text{mask})$  indicates a shrink/expand operation, which removes one unit wide features. Finally, a superscript on the mask name indicates the kind of rasterization to do. If NP represents the polysilicon mask, then  $NP^+$  specifies rasterization for a width check and  $NP^-$  specifies rasterization for a spacing check.

#### 4. Boundary Checking

So far we have seen how to shrink and expand masks and perform width checks on them. Another type of design rule depends on the relative positions of the edges of two masks. An example of this is the transistor extension rule, which requires polysilicon or diffusion to extend beyond the edges of transistor gate areas. This section starts by giving examples of boundary checks required by the Mead/Conway design rules.<sup>8</sup> Then an architecture is described for a custom chip that implements these checks. Finally, a notation is developed for specifying boundary checks.

#### 4.1 Boundary Check Examples

In order to check design rules which involve boundary interactions, it is necessary to use a 2x2 window to look at two masks simultaneously. Figure 16 shows three examples of boundary errors involving the polysilicon and diffusion masks. For each case, one or more 2x2 windows are shown which identify the presence of the error. A capital P or D indicates that the specified mask must be present at that position, a lowercase letter indicates that the mask must be absent, and a dash represents a don't-care position.

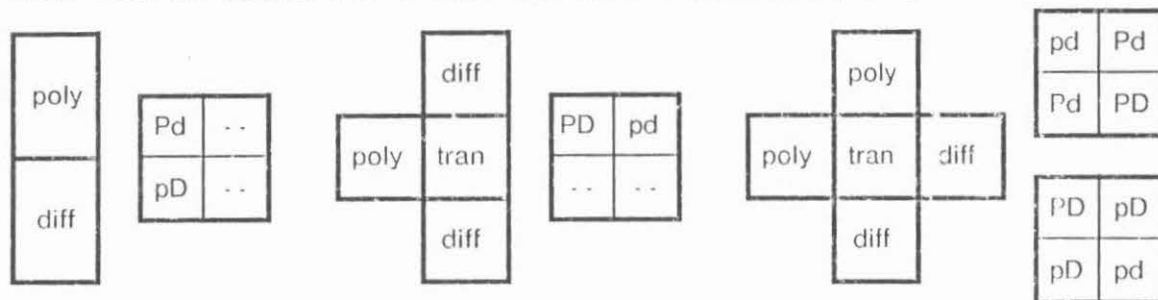


Figure 16: Three Boundary Check Examples

The first boundary error involves the polysilicon/diffusion spacing rule, which requires a  $1\lambda$  separation except where they cross to form a transistor. This rule cannot be completely checked by a width test on the union of the two masks because they might touch, as shown in the lefthand entry in figure 16. This error occurs whenever a raster cell which contains polysilicon but not diffusion is next to a cell that contains diffusion but not polysilicon. The window that is shown is one of four rotations that must be checked. Note that it is not necessary to check whether the two masks touch at a corner because that case is detected by a width test on the union of the two masks.

The center entry in figure 16 shows a case where the poly mask fails to extend beyond the edge of a transistor gate area. As the window illustrates, this error can be found by looking for a raster cell that contains both polysilicon and diffusion next to a cell that contains neither polysilicon nor diffusion. Again, all four rotations must be checked.

The righthand entry shows a more subtle transistor error. Here, the extension rule is satisfied but the transistor is still incorrect. It is necessary to check that polysilicon and diffusion actually cross the transistor gate area. This can be done by looking at the corners of the transistor. The two righthand windows in figure 16 check for the errors found at the upper left and lower right corners of the transistor gate area. It is necessary to check all four rotations of each window.

#### 4.2 Boundary Check Chip

The structure of the boundary check chip is similar to that of the width check chip in that row inputs are fed through shift registers into a PLA which checks the input window for errors. Unlike width checks and shrink operations, there are too many different possible boundary checks to specify them all in advance. Since it is necessary to be able to program the boundary check chip for the specific check that is desired, a dynamically programmable logic array (DPLA) is used instead of the mask programmable logic array used in the width

check/shrink chip. The window patterns programmed into the DPLA are stored in dynamic nodes and must be refreshed periodically.

Figure 17 gives the structure of the boundary check chip. Two rows each from two separate masks are input to the chip. One bit of shift register is provided for each input row so that a 2x2 window on each mask is input to the DPLA. The LOAD SELECT lines control when data on the DATA BUS is used to program the DPLA, which has eight AND terms. The first four terms are OR'ed together to produce ERROR 1 and the other four are OR'ed together to produce ERROR 2. Each of the AND terms in the DPLA specifies a boundary error.

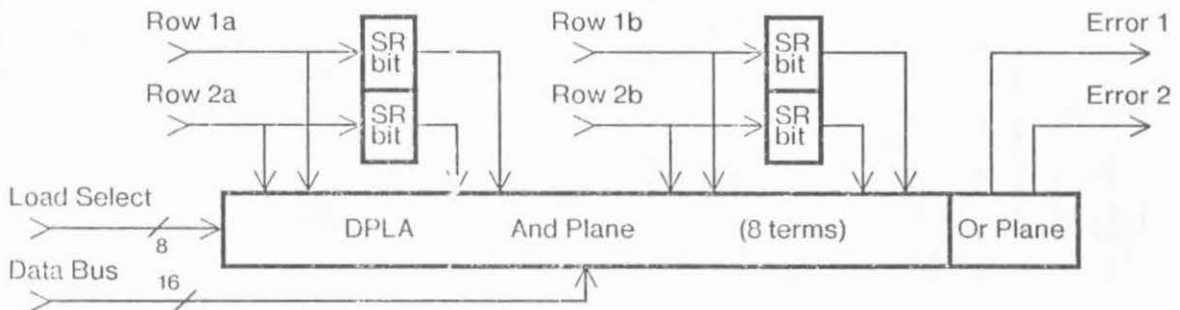


Figure 17: Structure of Boundary Check Chip

Normally, the boundary check chip would input two masks directly from the derived mask generator, but this is not necessary. Figure 18 illustrates an alternate way to use the boundary check chip that allows the inputs to have a shrink operation applied to them before being boundary checked. Note that type of shrink can be selected separately for each width check chip. This is especially useful when the grid size is small relative to the feature size, since many masks will have to be shrunk or expanded before they can be compared.

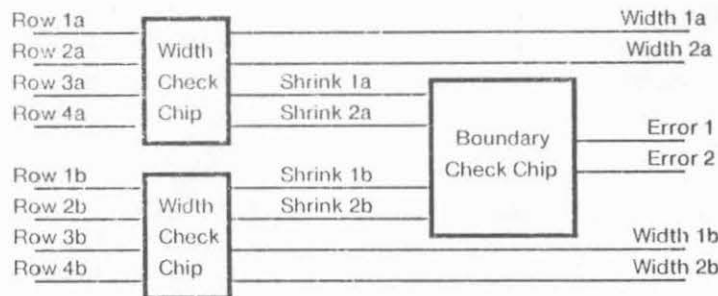


Figure 18: Boundary Check Chip with Pre-shrink

#### 4.3 Boundary Check Notation

A boundary check consists of a set of 2x2 window comparisons performed on two masks. If one of the window patterns is matched, then a boundary error has been found. The two masks are referred to as A and B; they are defined by equations involving shrinks, expands, and boolean operations on masks input from the rasterization unit. Each window is defined by a conjunction of subscripted terms, where each term is a dash or an upper or lower case A or B. The subscript defines the position of each term in the 2x2 window and

the choice of character indicates whether mask A or B is required to be present or absent at that position. For example, if A and B represent the polysilicon and diffusion masks respectively, the leftmost window in figure 16 would be represented as  $A_{11}b_{11}a_{21}B_{21}$ .

When specifying a boundary check, it is usually necessary to check for several rotations or reflections of the basic window pattern. In the examples in figure 16, each window is one of four orthogonal rotations that must be checked. For clarity, a sequence of windows that are rotations or reflections of each other can be specified as 4[window] to indicate that the four rotations of the window should be checked or 8[window] to indicate that all eight rotations and reflections should be checked. The equations below use this notation to specify the three boundary checks in figure 16. Equation 4.1a is the fully expanded form of equation 4.1.

$$4.1 \quad A = NP, B = ND: 4[A_{11}b_{11}a_{21}B_{21}]; \quad \text{poly/diffusion spacing}$$

$$4.1a \quad A = NP, B = ND: A_{11}b_{11}a_{21}B_{21}, A_{12}b_{12}a_{11}B_{11}, A_{22}b_{22}a_{12}B_{12}, A_{21}b_{21}a_{22}B_{22}; \\ \text{poly/diffusion spacing}$$

$$4.2 \quad A = NP, B = ND: 4[A_{11}B_{11}a_{12}b_{12}]; \quad \text{transistor edge extension}$$

$$4.3 \quad A = NP, B = ND: 4[a_{11}b_{11}A_{12}b_{12}A_{21}b_{21}A_{22}B_{22}], 4[A_{11}B_{11}a_{12}B_{12}a_{21}B_{21}a_{22}b_{22}]; \\ \text{transistor corner extension}$$

## 5. Conclusions

This paper has presented a hardware architecture for a raster scan design rule checking algorithm. The input to the hardware is the set of intervals that are covered on each scan line for each mask. The output is a list of positions where each type of error was found. The hardware can handle a wide variety of design rules, layouts of arbitrary size with an arbitrary number of layers, and features with edges at 45° angles. The use of custom chips makes it small and inexpensive enough to include in individual designer workstations. Furthermore, the hardware is fast enough that the speed of the design rule check is limited by the speed at which the controlling processor can provide input.

Such a significant speed increase can greatly change the way in which design rule checking is used. At present, design rule checking is usually done, if at all, as a postprocessing step after a layout has been mostly completed. In university designs, the design rule check is often not performed at all. The availability of a really fast design rule checker makes it easier to check a layout during all phases of the design effort and not just near the end, when it is hardest to fix any errors that are found.

It should also be noted that the basic architecture of the DRC system is applicable to a variety of other problems. The rasterization unit could be used to drive raster scan printing devices or could be used to scan convert images for a display screen. With slight changes, the architecture could be used to speed up the inner loop of a raster scan node extraction algorithm.<sup>1</sup> By introducing a different set of primitive window operations, the architecture could also be used in image processing applications.

Now that the architecture has been defined, an effort is under way to lay out the four custom chips mentioned in this paper and build a prototype of the design rule check hardware. A paper in preparation will define several variations of the Mead/Conway design rules could be checked using the notations developed in this paper.<sup>10</sup>

## Acknowledgements

Many thanks are due to Jon Allen, Clark Baker, Lance Glasser, Gary Kopec, Paul Penfield, and Chris Terman for their ideas and encouragement. Jon Allen deserves special thanks for his support, in all senses of the word.

## References

- [1] Baker, C.M., Terman, T., "Tools for Verifying Integrated Circuit Designs," *Lambda: the Magazine of VLSI Design*, Volume 1, number 3, Fourth Quarter, 1980
- [2] Baker, C.M., Massachusetts Institute of Technology, private communication, December 1980
- [3] Baird, H.S., "Fast Algorithms for LSI Artwork Analysis," *Proceedings of the 14th Design Automation Conference*, pp. 303-311, June, 1977
- [4] Bently, J.L., Haken, D., Hon, P.W., *Statistics on VLSI Designs*, CMU-CS-80-111, Department of Computer Science, Carnegie-Mellon University, April, 1980
- [5] Dunn, W., General Instrument Corporation, private communication, November 1980
- [6] Locanthi, B., "Object Oriented Raster Displays," *Proceedings of the Caltech Conference on Very Large Scale Integration*, pp. 215-225, January, 1979
- [7] McGrath, E.J., Whitney, T., "Design Integrity and Immunity Checking: A New Look at Layout Verification and Design Rule Checking," *Proceedings of the 17th Design Automation Conference*, Minneapolis, pp. 263-268, June, 1980
- [8] Mead, C., Conway, L., *Introduction to VLSI Systems*, Addison-Wesley, Massachusetts, 1980, pp. 47-51
- [9] Rosenberg, L.M., "The Evolution of Design Automation to Meet the Challenges of VLSI," *Proceedings of the 17th Design Automation Conference*, Minneapolis, pp. 3-11, June, 1980
- [10] Seiler, L., "Formal Definition of the Mead/Conway Design Rules," private communication, MIT VLSI Memo Series (in preparation)
- [11] Whitney, T., *A Hierarchical Design Rule Checker*, Master's Thesis, California Institute of Technology (in preparation)

# A VLSI TACTILE SENSING ARRAY COMPUTER<sup>1</sup>

John E. Tanner

California Institute of Technology

Marc H. Raibert<sup>2</sup> and Raymond Eskenazi

Jet Propulsion Laboratory

## ABSTRACT

Here we describe a device that is at once a special purpose parallel computer and a high resolution tactile array sensor. We are interested in extending the technology that gives a robot manipulation system information about contact between the manipulator hand and its environment. We have replaced the passive substrates of earlier tactile sensors with a custom designed LSI device that handles transduction, computing, and communication. Large metal electrodes on the surface of the device are placed in contact with a conductive rubber. Deformations of this elastic material are sensed by measuring changes in its local resistivity. The sensory architecture eases the problem of connecting the transducer and computer by using an array of processors to filter and reduce raw data before communication. Since large arrays covering an entire intact wafer are planned, the design includes backup redundancy for the computing elements, and mechanisms for automatic replacement of failed elements.

---

1. This paper presents the results of one phase of research performed at the Jet Propulsion Laboratory, California Institute of Technology, sponsored by the National Aeronautics and Space Administration under Contract NAS7-100.

2. Current address is Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa. 15213.



## INTRODUCTION

Versatile computer controlled manipulation depends heavily upon the availability of precise data generated from interactions between the manipulator and its environment, and upon computational elements that convert these data into useful control information. The sense of touch is a potential source of such data, but existing tactile sensing technology falls far short of our requirements. Furthermore, techniques for mating transducer with computer -- techniques that are essential to the ultimate success of man made sensory devices -- have matured quite slowly.

Low resolution tactile sensors have been made using an array of surface electrodes on a passive substrate covered with a pressure sensitive elastic material [Bejczy 78, Briot 79]. Large, dense arrays were not feasible with those methods, not because small electrodes were hard to make, but because a separate wire connected each electrode to the sensor electronics. The package of sensor electronics itself tended to be bulky, making it hard to use near the business end of a manipulator. Furthermore, the electronics only transformed and multiplexed raw data, leaving filtering, interpretation, and recognition to be accomplished elsewhere.

Here we describe a novel design that uses custom microelectronics to overcome these problems. We are constructing a high resolution tactile sensing array by designing a special VLSI circuit that performs three basic functions:

### Transduction -

The exposed surface of the circuit contains sets of electrode pads, much like bonding pads, that make direct contact with a layer of pressure sensitive elastic material. The surface nature of this interaction is key to the integration of sensor and computer. It attempts to overcome the pin limitation problem that usually limits parallel computing in integrated circuits by using a two-dimensional array of inputs.

### Computation -

Circuits in the device form an array of computational elements, one associated with each set of surface electrodes. Each element performs simple arithmetic operations and local communication functions with neighbors, while together they form a versatile parallel 'image' processor that performs discrete two-dimensional (2D) convolutions.



Communication - Use of a distributed shift register allows all outputs from the device to be communicated over a single wire. Repeated operation of the shift register causes the entire state of the sensor to be transmitted over this compact channel.

Our strategy in using tactile array data is to treat them as images, and to apply the processing techniques that have proven useful in dealing with other types of images, especially visual images. A number of such techniques developed by workers in computer vision rely on computing the 2D convolution between an image and sets of pre-specified feature masks. Such a convolution is given by:

$$C(x,y) = \sum_{i=1}^N \sum_{j=1}^N P(x+i-1,y+j-1) M(i,j)$$

where  $P(x,y)$  is the value of pressure on a cell at  $(x,y)$ , and  $M(i,j)$  is the corresponding value in an  $N \times N$  filtering mask. To perform this calculation it is necessary to index the transduced data, find products, and sum the results. Such convolutions with appropriate masks permit identification and location of lines, edges, and other contours important to economical descriptions of the objects found in an image [Davis 75].

An important feature of convolution is that it can be efficiently implemented by an array of processors, each performing the same calculation as every other processor in the array, all operating simultaneously, and each only requiring data from its own local neighborhood of the image. Therefore we have implemented an array processor that performs 2D convolution between programmable masks and tactile images.

## WHAT WE DID

Figure 1 illustrates the physical layout of the tactile sensor. An array of electrodes is covered by a sheet of pressure sensitive conductive rubber. This elastic material has the property that deformation causes its sheet resistivity to change in a predictable fashion. By passing a small test current from a pair of surface electrodes through the material, the magnitude of surface pressure can be measured locally.

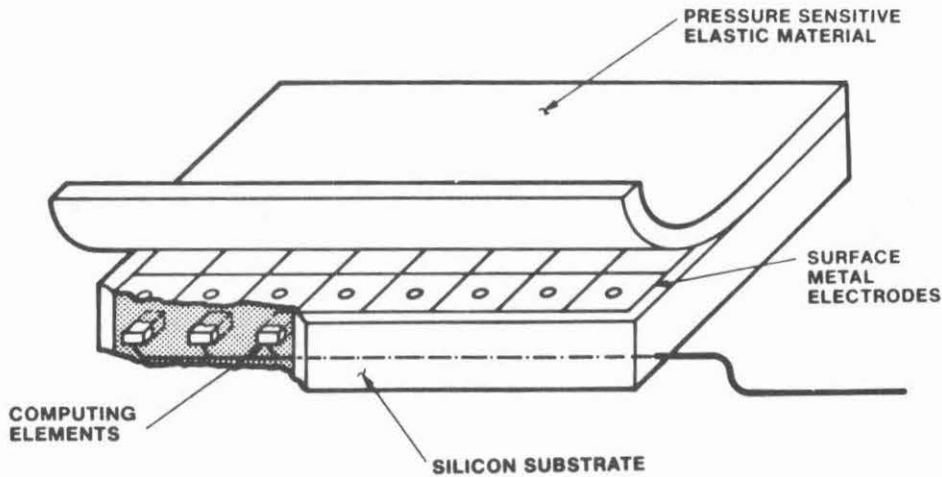


Figure 1. Mechanical architecture of touch sensor. A layer of pressure sensitive rubber is placed in contact with VLSI wafer. The surface of the wafer is covered with large sensing electrodes that are connected to circuitry that makes up an array processor within the wafer. Communication with the device takes place over a few serial lines.

In previous implementations the electrodes shown in Figure 1 were fabricated on a passive epoxy or ceramic substrate used only for mechanical support [Bejczy 78]. In our design, however, a surface layer of metal is placed on a silicon wafer to form these electrodes, while conventional nMOS circuitry below a protective glass layer forms the computing elements needed to implement transduction, filtering, data reduction, and communication. We have created a sandwich of conductive rubber, metal, and processed silicon. This is shown in Figure 2.

Figure 3 shows the overall architecture of the tactile sensing computer. An array of cells that transduce and compute are each connected to their nearest neighbors and to a global control bus. This global bus is driven externally to provide power, clock, instructions, and voltage reference to the elements of the array. The tactile cells either sense pressure, compute or communicate, with the whole array performing in lockstep under external control.

The computing units are simple but powerful processors that participate cooperatively to implement the algorithms required for tactile image sensing and analysis. Figure 4 shows a block diagram of a computing unit. Each unit contains its own set of transduction

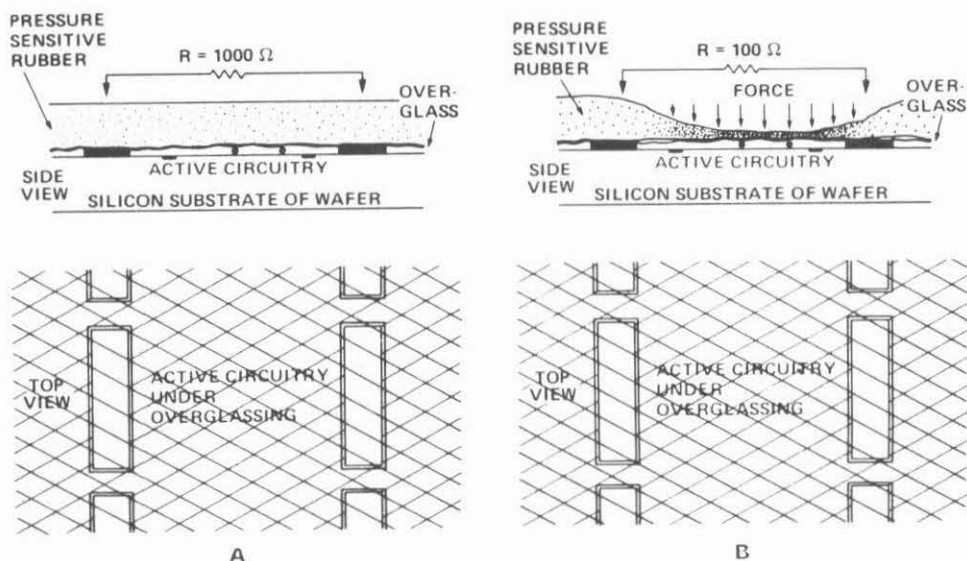


Figure 2. A) The sheet resistivity of the conductive elastic material is measured locally by metal electrodes that make contact through cuts in the overglass. B) When the material is compressed, carbon particles become more densely packed, decreasing resistivity.

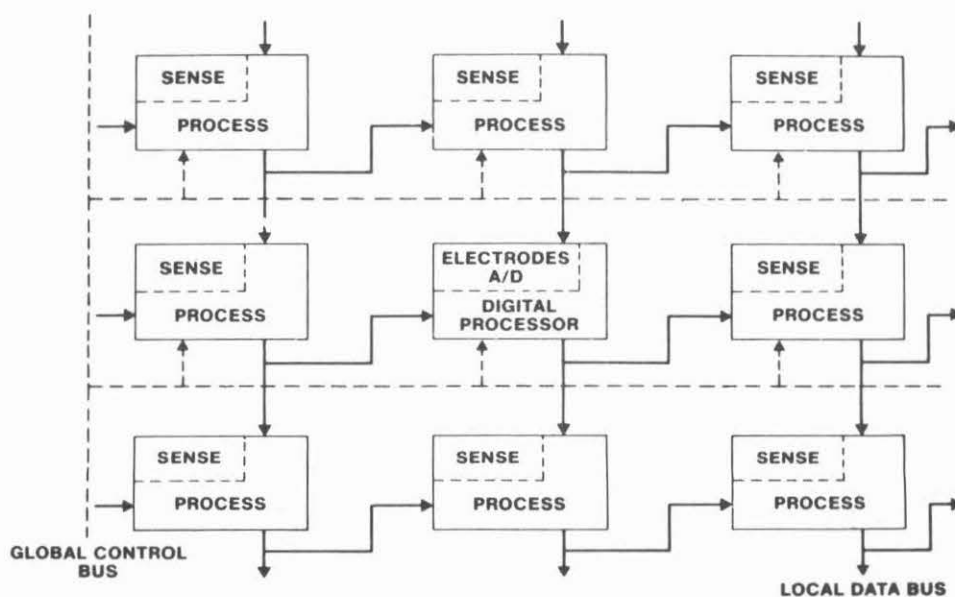


Figure 3. Block diagram of array processor. Each tactile cell has a sensing part and a computing part. A global control bus provides all cells with power, synchronization signals, and instructions. Cells are locally connected to nearest neighbors.

electrodes, an analog to digital converter, a latch, a simple arithmetic logic unit and an instruction register. The analog part converts the variable resistance of the conductive rubber into a 1-bit digital value that corresponds to the pressure on the cell. An adjustable global reference voltage allows the threshold of this analog-to-digital conversion to be varied. The result is stored locally in a latch, and can, under external program control, be made available to any of the four nearest neighbors. The latched data can also be multiplied by a number obtained from the global control bus, with the result accumulated in a 6-bit register using two's complement arithmetic. The contents of this 6-bit accumulator can be shifted and rotated in various ways.

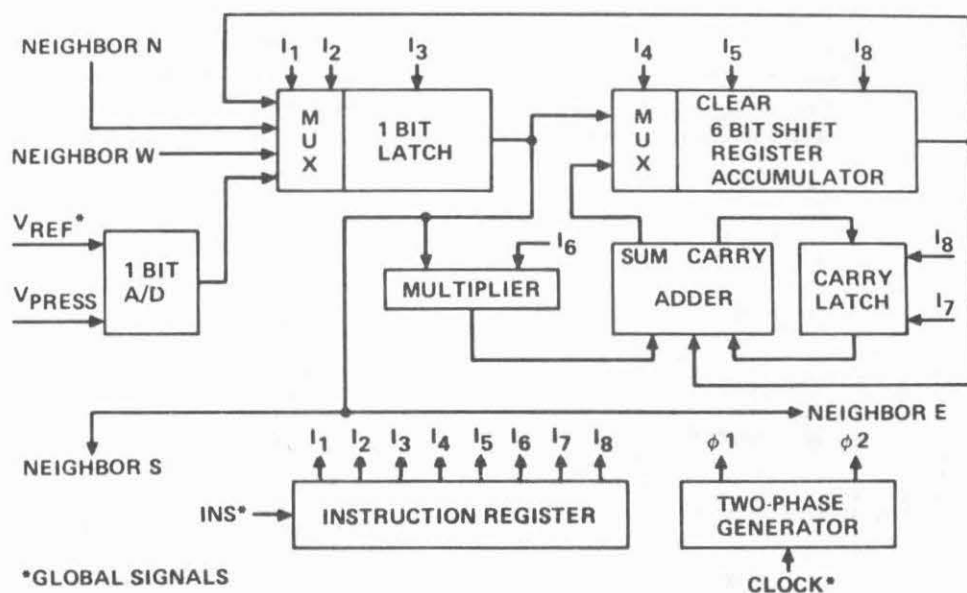


Figure 4. Block diagram of computing element.

The instruction register controls all operations of the computing element. The eight bits of the instruction are transmitted serially over a single global instruction line to the instruction register of each cell in the array. Eight instruction lines, one for each bit in the instruction register, control the various parts of the computing unit. Appendix A lists the functions of the instruction lines and gives the mnemonics and coding for the instruction set.

The computing resources just described allow the cell to sample the local pressure, to store the value, to pass data to neighboring cells, and to perform computations on the data that implement a 2D convolution with programmable mask. A simple slip detection algorithm is also easily implemented using these computational abilities. Example programs using the instruction set of Appendix A to do convolution and slip detection are given in Appendix B.

## EXTENSIONS TO LARGE ARRAYS

The ultimate target of this project is to construct a 30 by 60 mm. array of  $1 \text{ mm}^2$  tactile elements. This involves a very large area of active silicon compared to conventional designs, and therefore a much larger risk of fabricating defective circuitry. Normally, integrated circuits are manufactured by building an array of identical circuits on a silicon wafer which is subsequently diced into "chips" that are packaged and used separately. Since each chip is used separately, defective chips can be discarded on an individual basis in a straightforward manner. The overall yield can be kept high despite the presence of defective circuitry on the wafer.

Using today's fabrication techniques, a fair number of defects must be expected on a device that is  $1800 \text{ mm}^2$ . The present scheme, however, requires that all tactile sensing elements function correctly for the complete tactile sensor to operate effectively. Therefore, measures must be taken to eliminate the effects of these defects. Our designs provide a spare computing element for each tactile sensing element -- all sensing and computing circuits are replicated within the tactile cell (Figure 5). A selector circuit chooses between the pair of redundant computing elements.

We have designed a very simple 12 transistor selector that replaces a failed computing element with its backup spare. It is vitally important that this selector element be simple, since no backup is provided for the selector itself. Making it simple reduces its area and, therefore, the probability of it having a fabrication defect.

The selector is just a latch that changes state whenever its two inputs, both signals from the primary computing element, are not the same during the selector's strobe pulse. One input is the transducer output. The other is the computing element's output line. All sections of the primary computing element can be checked for proper operation by



For an array with  $N$  tactile elements the array yield will be:

$$\text{Yield} = P^N = e^{-ADN}$$

If each tactile cell contains duplicate computing elements and a selector the probability  $P_R$  that any one tactile cell is good becomes:

$$P_R = P_S - P_S(1-P)^2 + (1-P_S)P$$

where  $P_S$  is the probability a selector is good. Here we have assumed that a failed selector chooses one or the other of the computing elements it controls, rather than choosing neither or both. Also, the reliability of the global data bus is not considered. The yield of an array of  $N$  tactile cells, each with one backup computing element is then:

$$\text{Yield} = P_R^N$$

Figure 6 plots expected yield as a function of array size with defect density of  $0.05/\text{mm}^2$ , selector area of  $0.045 \text{ mm}^2$ , and computing element area of  $0.90 \text{ mm}^2$  for both redundant and non-redundant cells. It can be seen from these plots that arrays containing 1000 elements are achievable.

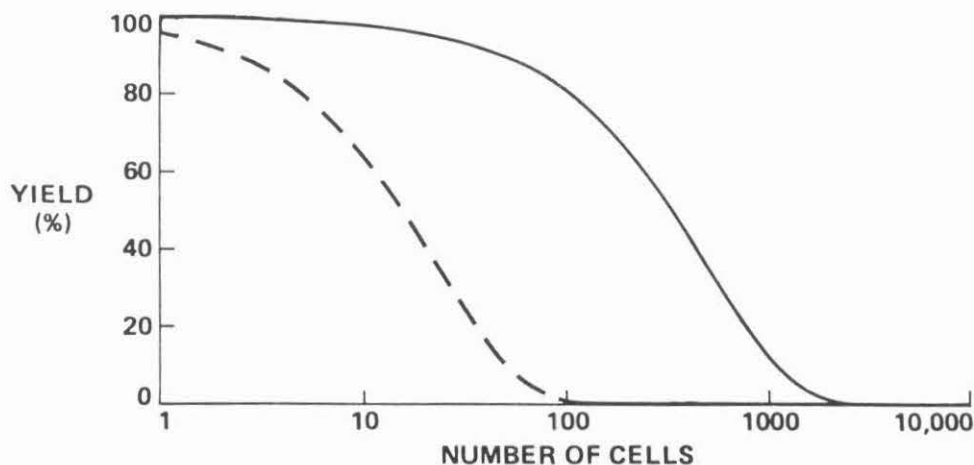


Figure 6. The expected yield of a tactile sensing array is calculated using a Poisson model of defect distribution. It is plotted here as a function of  $N$ , the number of cells in the array. The dotted curve is the yield with no redundancy. The solid curve is the yield with one spare computing element per cell as described in text. Defect density is  $0.05/\text{mm}^2$ , selector area is  $0.045\text{mm}^2$  and computing element area is  $0.9\text{mm}^2$ .

Figure 7 is a floorplan and photograph of a chip that implements a 1x2 tactile sensing array computer. In order to see the active circuitry the pressure sensitive material is not present in this photograph. The chip includes two tactile cells, each of which has two computing elements and a selector. Bonding wires are attached to conventional input/output pads along the top and bottom. These are used to test this early prototype. The large metal areas in the center of each tactile cell are the sensing electrodes. The size of this chip is about 2mm x 3mm.

This first prototype was fabricated as part of MPC5-80, the Multi-Project Chip effort coordinated by the Xerox Palo Alto Research Center [Conway 81]. A 3x3 array was fabricated as part of MOSIS, a similar service coordinated by ISI [Cohen 81]. Both versions are nMOS with  $\lambda$  equal to 2.5 micron. Table 1 gives the measured performance of the 3x3 tactile sensing array.

Pressure Sensitivity	50 grams
Clock Rate	3 MHz
Instruction Time	3 $\mu$ sec
6 Bit Add (Multiply)	18 $\mu$ sec
3x2 Convolution	140 $\mu$ sec
Power Requirements	5 V
	10 mA/Cell

TABLE 1. PERFORMANCE OF TACTILE SENSING CHIP

## PROBLEMS AND PLANS

An important problem with our present nMOS implementation is the excessive power requirement for circuits with large active areas. For small arrays, the current of 10 mA/cell is reasonable, but an 1800 element array, our original target, would require 18 amps. We are designing a new version of the tactile sensor in CMOS. This fabrication technology is expected to become available to us soon through the ISI implementation system. A CMOS design should greatly relieve the power supply and dissipation problems inherent in nMOS.



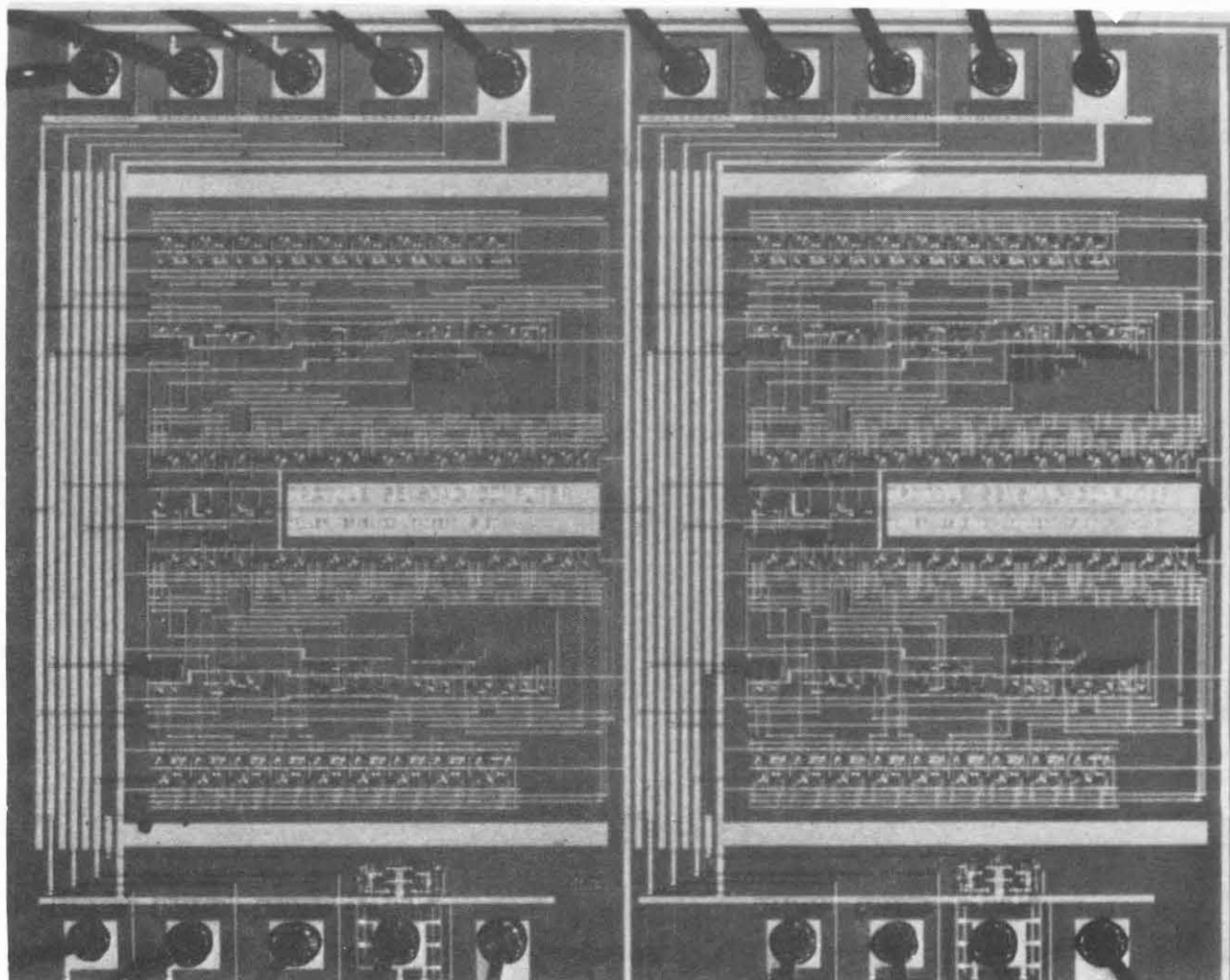
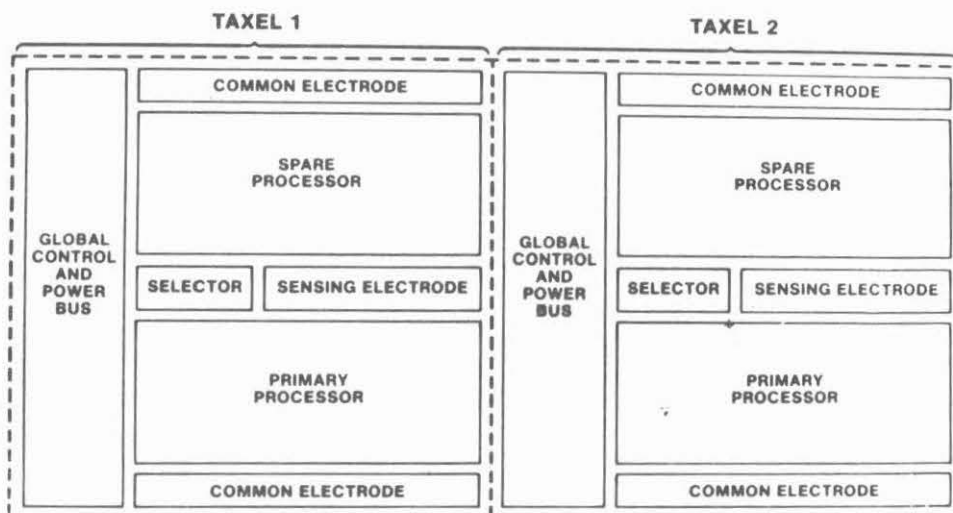


Figure 7. Floorplan and photograph of a tactile sensing chip produced by MPC5-80.

Several variations on the basic transduction scheme are being considered for future implementation. One of these uses a layer of the fabrication process as a piezoresistive transducer. This layer could be a custom layer added after the circuit has been fabricated conventionally. It may be possible to utilize the polysilicon layer as the pressure sensitive element, since it typically exhibits a slight piezoresistive effect. This technique is used in some pressure transducers available commercially. Another possible method of detecting pressure is to use a layer of material that changes its optical properties with pressure. The circuitry below must then detect the optical difference. The advantage of this technique is that it allows a greater isolation of the integrated circuitry from the physical and chemical dangers of the environment.

We also hope to develop other methods of fault tolerance. The information content of an image is degraded only slightly by the loss of one pixel. In the tactile sensing array if one cell fails, all the data from other cells that are to be shifted through the bad cell are lost. This results in an image with a stripe defect. We are developing a more complex shifting pattern that routes each cell's information toward the edge of the array through multiple pathways. This new design requires the ALU within each cell to reconstruct correct values periodically in the shifting pattern by computing the majority function of the values that arrive over different paths. Simulations show that this method of redundant communication gives an array a 40 fold increase in immunity to stripe defects.

Combining transduction with electronic circuitry in an intimate way gives rise to a set of problems unique to this application of VLSI. The transducer must be in contact with the environment it is sensing, in this case, the gripping surface of a robot's hand. The electronics, which in most applications are protected from the environment by a sturdy sealed package, is now an integral part of the transducer. The silicon substrate must be able to withstand any force that the robot's hand is expected to encounter and to resist any chemical contamination diffusing through the pressure sensitive elastic covering. In addition, heat sinking must be provided to the substrate to remove the heat produced by such a large area of active circuitry. Further testing will show how critical these packaging problems are to producing a practical, useful tactile sensor.

So far we have not thoroughly characterized the sensor's analog response to forces and distortions. This must be done. The several different types of available pressure sensitive elastic materials will be tested with a special electrode chip -- one that has six different electrode geometries. Sensitivity, range, and localization are the variables of interest.

## SUMMARY

Combining transduction with computation in the same device is an effective way to use the power of VLSI technology, and to overcome the traditional sensing problems of interconnection, communication and computation. The two-dimensional nature of the integrated circuit nicely matches the surface nature of the tactile sensing problem and the architecture of array processors. The convolution algorithm used to refine raw image data exploits the full concurrency of an array processor to achieve high performance. Wafer scale integration is necessary to build working arrays of useful size, and fault tolerant techniques are necessary to achieve wafer scale integration. These ideas have been brought together to start a new generation of tactile sensors for robots.

## ACKNOWLEDGEMENTS

The authors wish to thank Glen Okita and Dean Uehara for many early contributions to this work.

## REFERENCES

- Bejczy, A. K., "Smart Sensors for Smart Hands," AIAA/NASA Conference on "Smart Sensors," November 1978, Hampton, VA.
- Briot, M., "The Utilization of an Artificial Skin Sensor for the Identification of Solid Objects," 9th International Symposium on Industrial Robots, Washington, D.C., March 1979.
- Cohen, D., "MOSIS -- The ARPA Silicon Broker," Proceedings of the Caltech Conference on VLSI, January 1981.
- Conway, L. A., "The MPC Adventures: Experiences with VLSI Implementation Systems," Proceedings of the Caltech Conference on VLSI, January 1981.
- Davis, L. S., "A Survey of Edge Detection Techniques," Computer Graphics and Image Processing, 1975, 4, pp. 248-270.

## Appendix A

The eight bits of the instruction word control different parts of the cell and can be identified as follows:

- $I_1$  Latch input select 0
- $I_2$  Latch input select 1
- $I_3$  Latch shift/store
- $I_4$  Accumulator MSB select
- $I_5$  Accumulator clear
- $I_6$  Multiplier value
- $I_7$  Carry Clear
- $I_8$  Accumulator and Carry latch shift/store

Listed below are some useful instructions and their mnemonics.

	$I_1$	$I_2$	$I_3$	$I_4$	$I_5$	$I_6$	$I_7$	$I_8$	(x = don't care)
CLAR	x	x	0	x	1	x	1	0	Clear accumulator and carry latch
CLRA	x	x	0	x	1	x	0	0	Clear accumulator only
CLRC	x	x	0	x	0	x	1	0	Clear carry latch only
SHR S	0	1	1	x	0	x	0	0	Shift all latch values one cell south
SHR W	1	0	1	x	0	x	0	0	Shift all latch values one cell west
STADC	1	1	1	x	0	x	0	0	Store the value from the a/d converter in the latch
ADDM1	x	x	0	0	0	1	0	1	Multiply latch value by 1 and add to accumulator LSB
ADDM0	x	x	0	0	0	0	0	1	Multiply latch value by 0 and add to accumulator LSB
ROTA1	0	0	1	1	0	x	0	1	Rotate the accumulator through the latch
ETCHA	x	x	0	1	0	x	0	1	Move the value in the latch into the accumulator MSB
NOP	x	x	0	x	0	x	0	0	No operation

To multiply the value in the latch by a mask value 5 and add the result to the six bit accumulator requires this sequence.

```

ADDM1
ADDM0
ADDM1      5 = 000101(binary)
ADDM0      •
ADDM0
ADDM0

```

## Appendix B

One possible mask for detecting vertical edges is shown in the example below. The pressure data show that an object with a rectangular corner is pressing on the lower left part of a 6 x 6 array of cells. The results of the convolution are 36 values, one computed by each cell, as shown on the right. The non-zero values indicate a vertical edge.

Pressure						Mask		After Filtering					
0	0	0	0	0	0			0	0	0	0	0	0
0	0	0	0	0	0			0	0	0	0	0	0
0	0	0	0	0	0	5	-5	0	0	0	0	0	0
1	1	1	0	0	0	5	-5	=>	0	0	5	0	0
1	1	1	0	0	0	5	-5		0	0	10	0	0
1	1	1	0	0	0				0	0	15	0	0
1	1	1	0	0	0				0	0	15	0	0

Figure B1. Example of edge detection using convolution.

Listed below is a program written using the mnemonics of Appendix A to perform the edge detection convolution of Figure B1.

INSTRUCTION	COMMENTS
CLFAR	NOTE: We are cell (1,1)
STADC	Get pressure data P(1,1)
ADDM1	M(1,1) = 5
ADDMO	
ADDM1	
ADDMO	
ADDMO	
ADDMO	
CLRC	
SHL1W	Get neighbor's data P(2,1)
ADDM1	M(2,1) = -5
ADDM1	
ADDMO	
ADDM1	
ADDM1	
ADDM1	
CLRC	
SHL1S	Get neighbor's data P(2,2)
ADDM1	M(2,2) = -5
ADDM1	
ADDMO	

... and so on for a total of 48 instructions

Figure B2. Program to perform 3x2 convolution.

Another computation that may prove useful is the determination of change in pressure from one moment to the next. This can indicate slip or be one step in a data compression scheme. Each cell can detect a change of pressure by computing the inequality function:

$$S = \begin{cases} 0 & , \text{ for } P(T_1) = P(T_2) \\ 1 & , \text{ for } P(T_1) \neq P(T_2) \end{cases}$$

where  $P(T_1)$  is the pressure on the cell at time  $T_1$  and  $P(T_2)$  is the pressure on the cell at a later time  $T_2$ . For single bit pressure values, the function  $S$  can be computed by adding  $P(T_1)$  to  $P(T_2)$ . The least significant bit of the result is  $S$ . The program listed in Figure B3 implements this algorithm.

INSTRUCTION	COMMENTS
CLEAR	
STADC	Get pressure data
ADDM1	Put it in accumulator LSB
ADDM0	(Multiply by 1)
ADDM0	
ADDM0	
ADDM0	
ADDM0	
STADC	Get new pressure data
ADDM1	Add it to accumulator
ADDM0	
ADDM0	
ADDM0	
ADDM0	
ADDM0	
ROTAT	LSB is 1 if slip occurred between samples

Figure B3. Program to calculate slip.





COMPUTER-AIDED DESIGN SESSION

*Chairperson: MARTIN NEWELL  
Member of Research Staff  
Palo Alto Research Center  
Xerox Corporation*



## Algorithmic Layout of Gate Macros<sup>\*</sup>

Daniel D. Gajski

Avinoam Bilgory

Joseph Luhukay

Department of Computer Science

University of Illinois at Urbana-Champaign

Urbana, Illinois 61801

The rapid advancement of VLSI technology necessitates new implementation methodologies with design automation capabilities. Existing implementation styles such as master slice, programmable logic arrays and custom design with cell library do not achieve the best tradeoffs between circuit density and chip development cycle time. The implementation methodology based on register-transfer building blocks called gate macros can be used to drastically cut down the design time. Furthermore, the gate macros which generally represent functional entities like registers, adders, busses, logic units etc. are subjective to algorithmic or totally automatic layout [Verg80], [Joha79].

This paper describes the basic modules of a gate-to-silicon compiler which accepts as its input a high level description of gate macros and generates a layout that satisfies particular technology (NMOS, for example) and environmental parameters (layout area or time delay, for example). The input to the gate-to-silicon compiler are the set of

---

<sup>\*</sup> This work was supported in part by the NSF under grant No. US NSF MCS80-01561

macros generated at the register transfer level. High-level language constructs like DO loops and IF statements are allowed in the input language. However, only Boolean scalars, vectors and strings are allowed. For example, a 16-bit binary adder can be described as follows:

```

S1:    C(0) = CIN
        DO I = 1,16
S2:    C(I) = A(I)*B(I) + (A(I) + B(I))*C(I-1)
S3:    S(I) = A(I) ⊕ B(I) ⊕ C(I-1)
        END
S4:    COUT = C(16)

```

The above description can be used for variety of implementation styles. For example, if the delay time specified is relatively slow with respect to technology used the 32-bit adder will be implemented as a ripple-carry adder. If a faster version is required the look-ahead-carry adder will be used. For different delay times different number of bits will be looked ahead. Similarly, different layouts will be produced for different time delays.

The compiler consists basically of four modules (Figure 1):

1. Boolean Analyzer partitions the input description into blocks with easily recognizable structure. For example, the statements  $S_1$  and  $S_2$  will be recognized as a recurrence system while the statement  $S_3$  is detected to be a vector operation. Statement  $S_4$  is detected as a scalar operation. Furthermore, the Boolean Analyzer generates the dependence graph with statements as vertices and dependences as edges. The dependence graph represents the internal structure of the gate macro. It

High Level Language description of gate macros

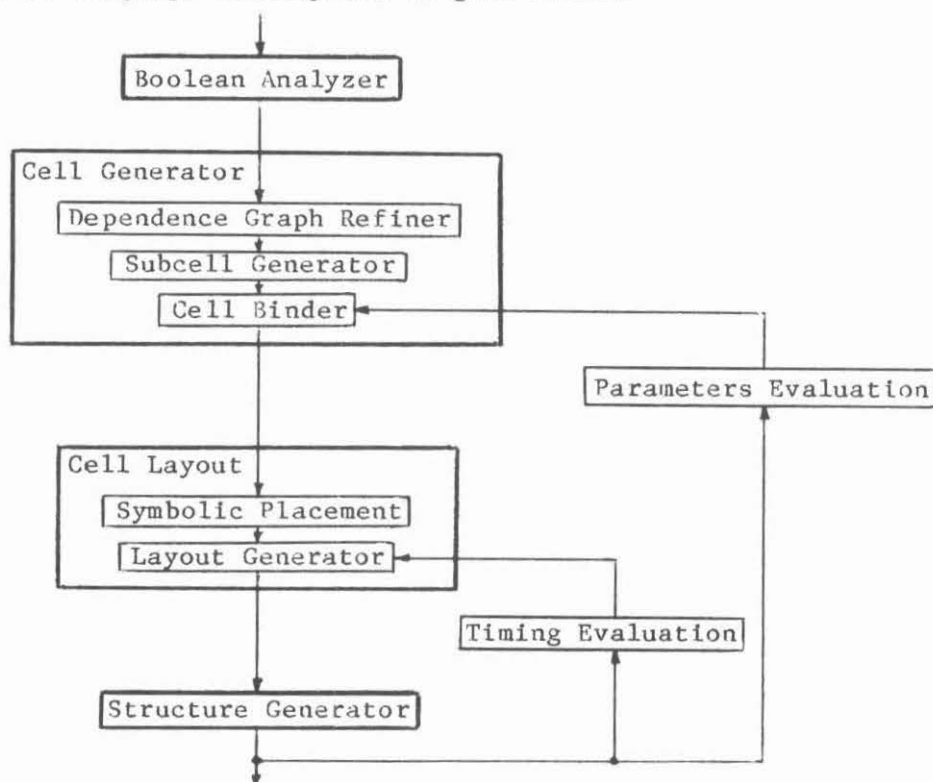


Figure 1. Block diagram of a gate-to-silicon compiler.

indicates the critical time delay and cell structure of the future layout alternatives.

2. Cell Generator modules consist of Dependence Graph Refiner, Subcell Generator and Cell Binder.

The Dependence Graph Refiner tries to break each of the dependence graph nodes into as many nodes as possible. The resulting dependence

graph is more detailed, which allows the Cell Binder more flexibility in optimization. Since statements  $S_1$  and  $S_4$  are scalar operations without operators their layout area and time delay are  $O(\epsilon)$  where  $\epsilon$  is a small value, so they are left untouched. Statement  $S_2$  is a recurrence with maximum  $O(n \log n)$  layout area and minimum  $O(\log n)$  time delay where  $n$  is the recurrence length. Since the recurrence node will be broken into three or more different types of subcells, its decomposition is left to the Subcell Generator. Statement  $S_3$  has an  $O(n)$  layout area and  $O(1)$  time delay. Since the EXCLUSIVE-OR operation is associative, statement  $S_3$  can be dissolved into  $S_{3a}$  and  $S_{3b}$ . Using the above approximation the original program is distributed as shown below.

```

S1:    C(0) = CIN
        DO I = 1,16
S2:    C(I) = A(I)*B(I) + (A(I) + B(I))*C(I-1)
        END
        DO I = 1,16
S3a:   T(I) = A(I) ⊕ B(I)
        END
        DO I = 1,16
S3b:   S(I) = T(I) ⊕ C(I-1)
        END
S4:    COUT = C(16)

```

The new dependence graph is shown in Figure 2.

The Subcell Generator consists of several submodules, each for one type of a block recognized by the Boolean Analyzer. Each submodule generates the functional description of the basic subcells used to synthesize the given block. The recurrence statements  $S_1$  and  $S_2$  generate

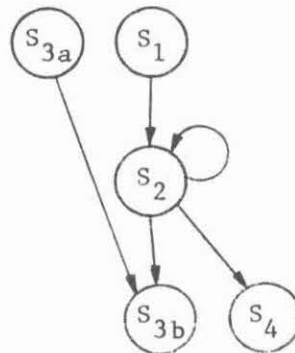


Figure 2. Dependence graph of distributed program.

four types of subcells:

type 2.1 subcell:  $G = A * B$

type 2.2 subcell:  $P = A + B$

type 2.3 subcell:  $G = G_1 + G_2 * P_1, \quad P = P_1 * P_2$

type 2.4 subcell:  $C = G + P * C_0$

A description of cell generation for recurrence structures is found in [BiGa80]. Statements  $S_{3a}$  and  $S_{3b}$  generate one type of subcell each, called type 3a and 3b subcells, respectively.

The Cell Binder combines subcells to form larger cells. The subcells to be combined are selected according to the constraints posed by the dependence graph. Since type 2.1 and 2.2 subcells (generated for the recurrence) perform vector operation as well as type 3a subcell, the three can be combined to form one cell called type 1 cell. Type 2.4 and 3b subcells can also be combined into one cell, but it was not done in this example, so type 2.3, 2.4 and 3b subcells will each be assigned one

type of cell and renamed as type 2, 3 and 4 cells, respectively. The layout occupies minimum area when all the cell types have similar widths. So, if the Structure Generator finds, for example, type 1 cell to be too large, a separate cell type may be dedicated to subcell 3a. Since  $S_{3a}$  is not on a critical path, this cell can be positioned almost anywhere in the layout in that case.

3. Cell Layout modules consist of Symbolic Placement and Layout Generator.

The Symbolic Placement module generates a two-dimensional array of symbolic transistors and their connections. Compaction is done automatically when this two-dimensional array is translated by the Layout Generator into a complete mask description in compliance with layout design rules of the chosen technology.

Each cell can be manually designed if so desired, leaving the placement and routing to be automatically performed by the system. The manual cell design presents one extreme of the provided layout design space [MeCo80]. However, the overall aim is to have an automatic layout system, where a manual cell design or a cell library is replaced by the library of algorithms in which one or more algorithms for automatic generation of layout specifications are available for each cell model supplied by the Cell Generator module. It then follows that the algorithmic layout is the other extreme of the layout design spectrum.

For example, an obvious approach would be to implement each cell with a small programmable logic array. The MOS and  $I^2L$  technologies are well adaptable to automatic synthesis as shown in [SOHT80] for one-dimensional gate arrays. We have chosen a two-dimensional array approach as described below.



The Symbolic Placement module is based upon a grid system of tracks - or channels - on different layers of the integrated circuit structure. Interaction among the layers is governed by the technology, and as a result, geometric relationship among the tracks is determined by the technology's layout design rules. Figure 3 shows a grid system which is used for silicon-gate MOS.

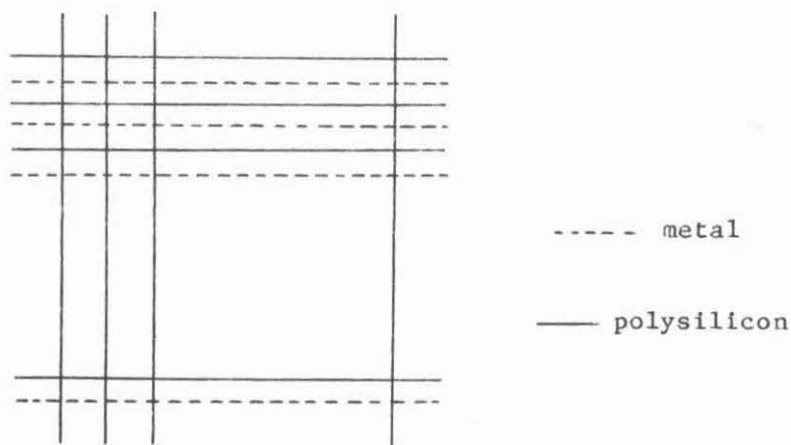


Figure 3. Sample grid system for MOS technology.

Here the metal layer is more or less independent of the polysilicon and the diffusion layers, whereas polysilicon and diffusion interact strongly with each other. Hence polysilicon and diffusion tracks can be "hidden" underneath metal tracks. Using this grid as a base, synthesis procedures have been developed. For example, using a metal and polysilicon grid like in Figure 3, two-dimensional arrays can be formed by manipulating the diffusion to form the necessary devices, interconnected such as to build the required circuit.

Figure 4 shows the processes implemented by the Cell Layout modules. Input to the Symbolic Placement module consists of functional description of a cell (or a set of cells), in the form of a set of AND-OR-INVERT Boolean equations. In addition to this, basic topological information about the cell is also given, which comprises assignment of topological attributes to the input-output nodes of the cell. For example, the cell shown in Figure 5(b) was specified with  $\bar{G}_1$ ,  $\bar{P}_1$  and  $\bar{T}$  (ordered from left to right) as top-inputs coming in polysilicon,  $\bar{G}_2$  and  $\bar{P}_2$  (ordered from top to bottom) as right-inputs coming in metal,  $G$ ,  $P$  and  $\bar{T}$  (ordered from left to right) as bottom-outputs going out in polysilicon, and  $G$  and  $P$  (ordered from top to bottom) as left-outputs going out in metal. The functional description specified for the cell is:  $G = \bar{G}_1 * \bar{P}_1 + \bar{G}_1 * \bar{G}_2$ ;  $P = \bar{P}_1 + \bar{P}_2$  and  $\bar{T} = \bar{T}$ .

If the I/O nodes ordering along the cell boundaries is fixed, such as in our case, then the Symbolic Placement module will start by ordering product-terms within an AND-OR-INVERT function, and also of the drive-transistors within a product-term. Otherwise, the module will first generate a symbolic placement of the functions themselves. The ordering's goal is to minimize the cell's height by reducing the number of horizontal tracks needed to lay out the cell. In our example, we need to place the product-terms of function  $G$  ( $\bar{G}_1 * \bar{P}_1$  and  $\bar{G}_1 * \bar{G}_2$ ), function  $P$  ( $\bar{P}_1$  and  $\bar{P}_2$ ) and function  $\bar{T}$  ( $\bar{T}$ ), such that  $\bar{G}_1$ ,  $\bar{P}_1$  and  $\bar{T}$  - which come in polysilicon - need not traverse any unnecessary vertical diffusion tracks. This is done by identifying the polysilicon input variable shared by both functions (here:  $\bar{P}_1$ ) and ordering the product terms such that metal crossovers for the polysilicon input variables (to get over diffusion tracks) are minimized. The following table shows how this process is done:

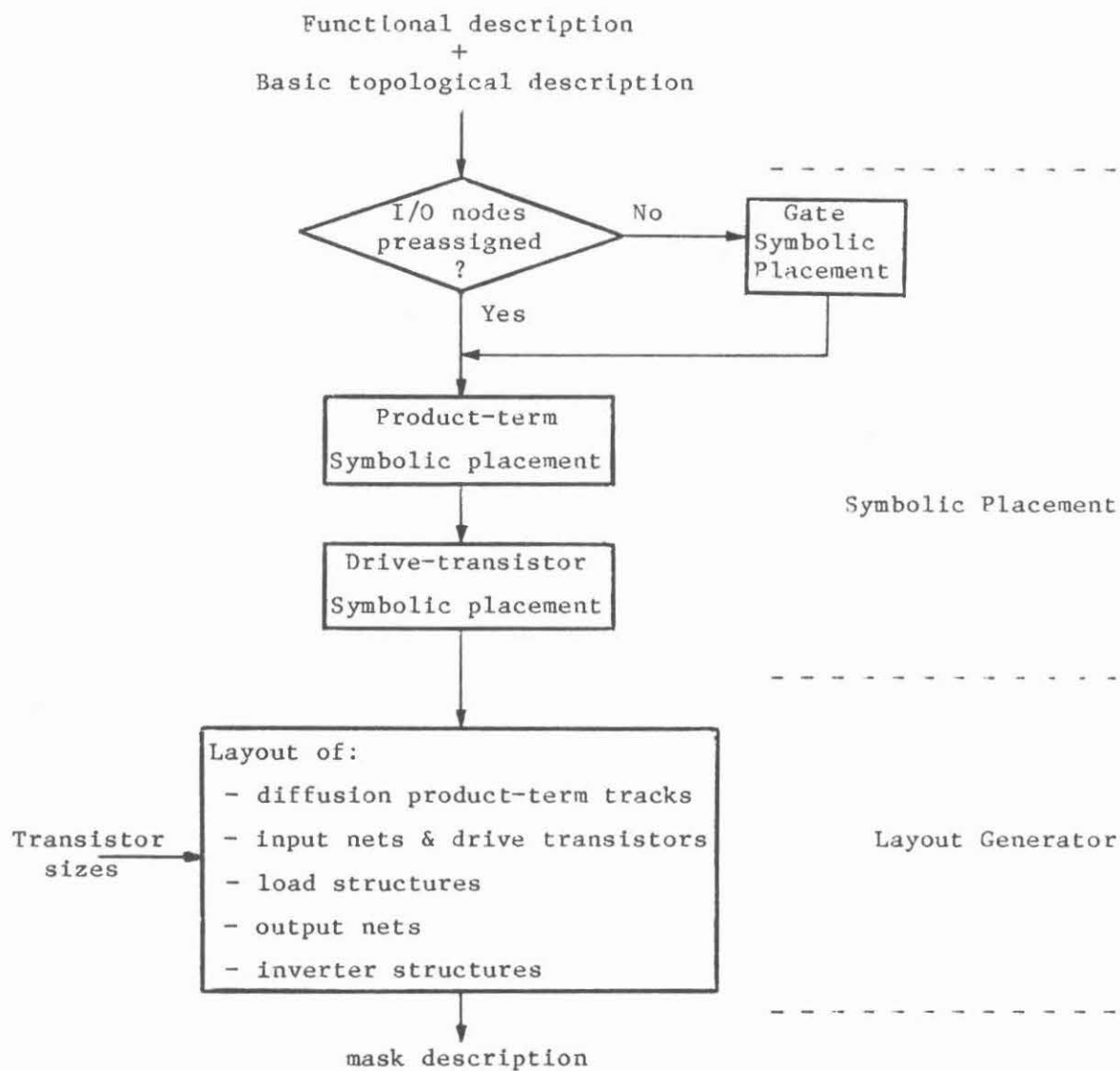


Figure 4. Block diagram of the Cell Layout modules.

		$\bar{G}_1$	$\bar{P}_1$	$\bar{T}$			$\bar{G}_1$	$\bar{P}_1$	$\bar{T}$
G:	$\bar{G}_1 * \bar{P}_1$	1	1	0	G:	$\bar{G}_1 * \bar{G}_2$	1	0	0
	$\bar{G}_1 * \bar{G}_2$	1	0	0		$\bar{G}_1 * \bar{P}_1$	1	1	0
P:	$\bar{P}_1$	0	1	0	P:	$\bar{P}_1$	0	1	0
	$\bar{P}_2$	0	0	0		$\bar{P}_2$	0	0	0
$\bar{T}$ :	$\bar{T}$	0	0	1	$\bar{T}$ :	$\bar{T}$	0	0	1
Before ordering					After ordering				

The output of the Symbolic Placement module is a table denoting relative placement of transistors on the reference grid system, and net-lists for the inputs and outputs. For our example, the table will be as follows:

$$\begin{array}{cccc}
 \bar{G}_1 & \bar{G}_1 & - & \bar{P}_2 \\
 \bar{G}_2 & \bar{P}_1 & \bar{P}_1 & -
 \end{array}$$

where columns denote vertical diffusion tracks, and rows denote horizontal polysilicon tracks.

The Layout Generator uses the symbolic placement data to generate the masks, described in an intermediate form like the CIF [MeCo80]. It generates the rectangles necessary to lay out the masks: diffusion product-term "tracks", input nets and drive transistors, load structures, output nets, and inverter structures. Figure 5 shows the simulated layout of four types of cells used in our example.

Rather than predefining device parameters and then laying them out using a placement and routing scheme, the circuits are first laid out in an array-like structure with minimum device sizes. The electrical and geometrical parameters are passed on to the next module. Iteration of

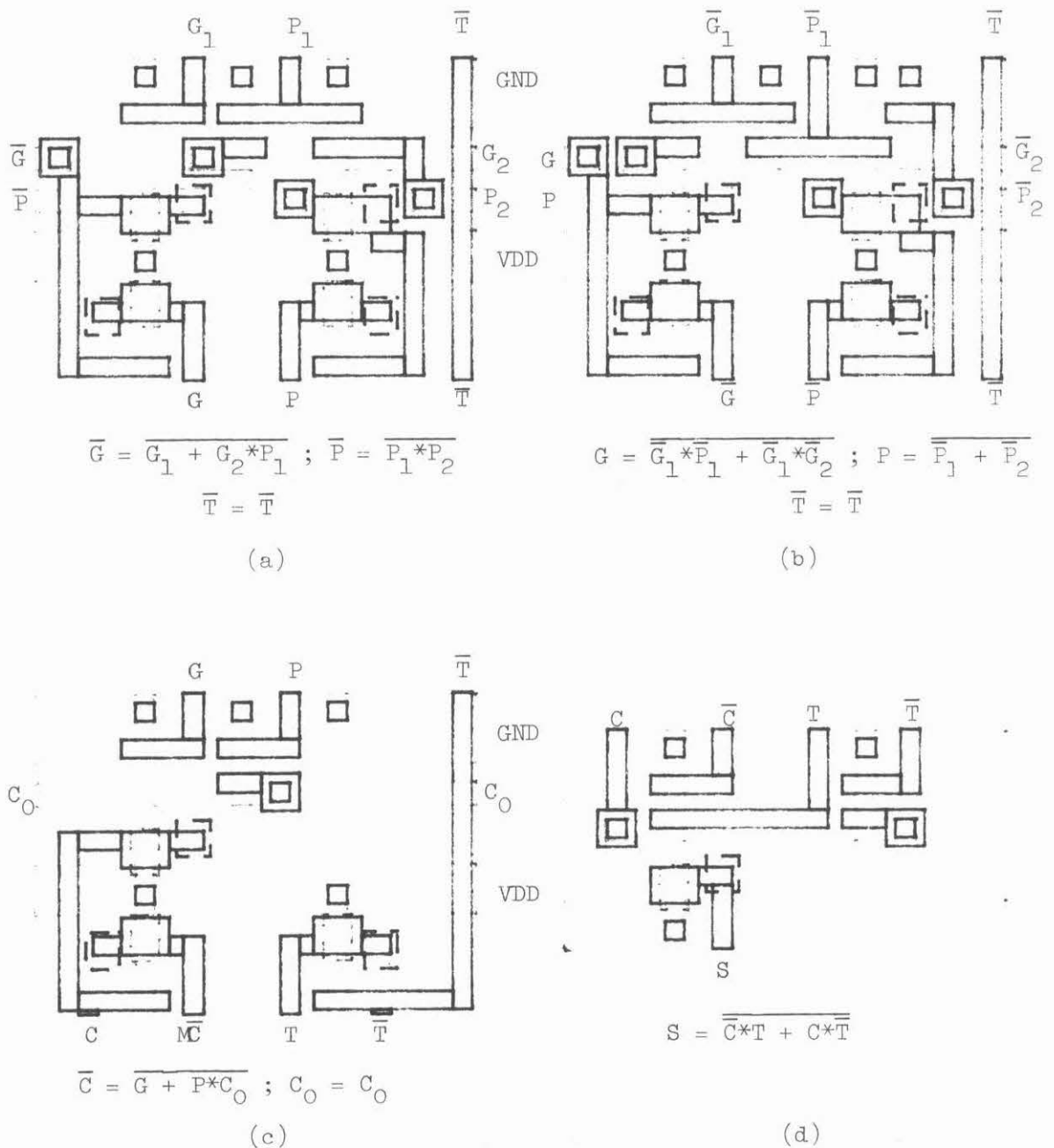


Figure 5. Layout of adder's basic cells:

- (a) Type 2a cell; (b) Type 2b cell;
- (c) Type 3 cell; (d) Type 4 cell.

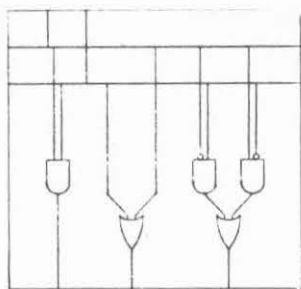
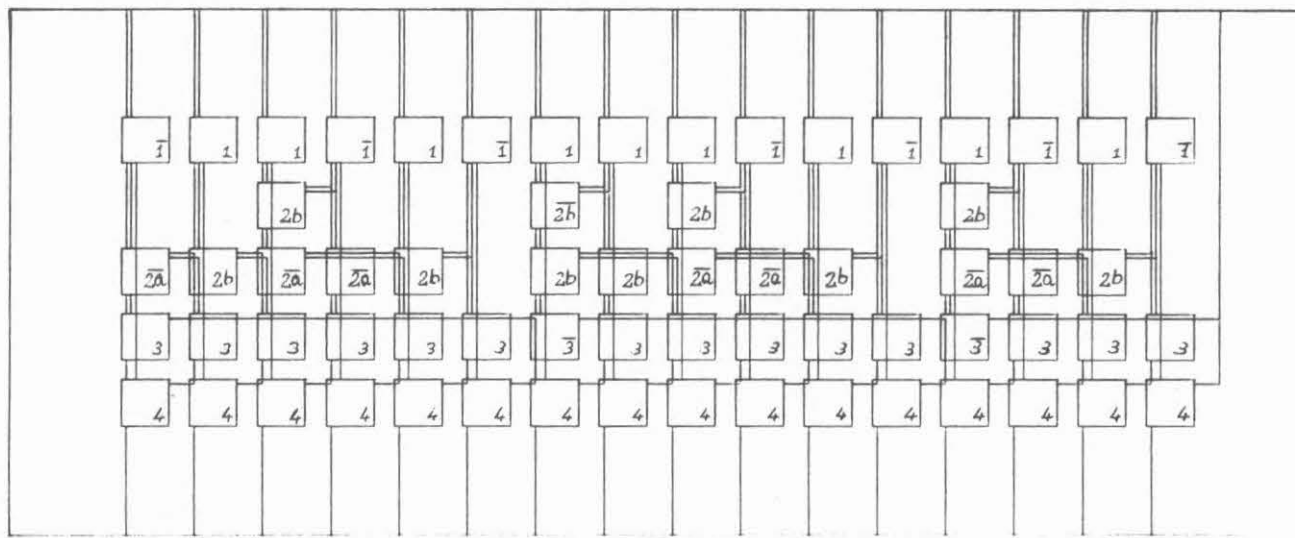
the process will produce the desired circuit with the device sizes necessary to meet the design goals.

4. Structure Generator attempts to obtain the best possible structure for the given functional description and environmental parameters. It specifies the cell types, the position of each cell in the final layout and the interconnections between the cells.

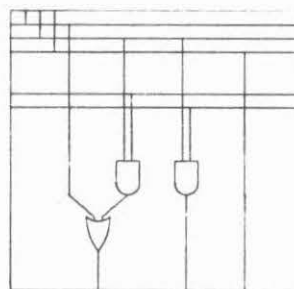
Figure 6 shows the structure of a 16-bit binary adder. Each cell will be referred to as  $C[i,j]$ , where  $i$  and  $j$  are the row and column where the cell is located, respectively, and the top rightmost cell is  $C[1,1]$ . Data are flowing only from top to bottom and from right to left. The four types of cells generated by the Cell Generator are located as follows: type 1 cells in the first row, type 2 in the second and third rows, type 3 in the fourth row and type 4 in the fifth row. The second, third and fourth rows perform the carry-look-ahead.

The input carry  $C(0)$  is fed into cells  $C[4,1]$  through  $C[4,4]$  which, together with the cells in the second and third rows above them, function as the carry-look-ahead for carries  $C(1)$  through  $C(4)$ . Then the output of cell  $C[4,4]$  (which is  $C(4)$ ) is fed into cells  $C[4,5]$  through  $C[4,10]$  that similarly produce the carries  $C(5)$  through  $C(10)$ . Lastly, the output of cell  $C[4,10]$  is fed into the cells to its left, so  $C(11)$  through  $C(16)$  are produced.

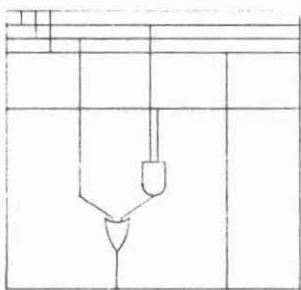
Let us assume that each type of cell produces its outputs in the same time delay  $d$  after its inputs are stable. For this particular adder example it was also given that the sum  $S(I)$  has to be available  $7d$  and the input carry  $C(0)$  is available  $3d$  after the inputs  $A(I)$  and  $B(I)$  are stable. Also, the fanout is limited: each cell can drive at most 7 other cells. The structure shown in Figure 6 meets these constraints



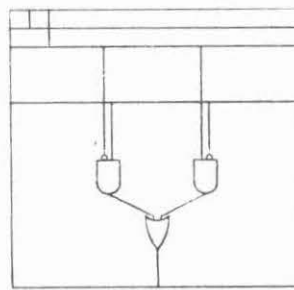
type 1 cell



type 2 cell



type 3 cell



type 4 cell

Figure 6. 16-bit adder structure and types 1, 2, 3 and 4 cells, in AND-OR form.

with a very important feature - it has the minimum number of rows, therefore it occupies minimum chip area (however, this structure is not unique).

Several paths through the structure have the maximum specified delay. They will be called critical paths (e.g  $C[1,6] \rightarrow C[3,6] \rightarrow C[3,8] \rightarrow C[3,10] \rightarrow C[4,10] \rightarrow C[4,12] \rightarrow C[5,14]$ ). The functions that define each type of cell are evaluated by the Cell Generator in a sum of products form. Since in MOS technology (where this example is implemented) an AND-OR-INVERT logic is implemented more naturally than AND-OR, the complemented outputs are produced by each cell rather than the true ones. Inverting the outputs again is ruled out, since it almost doubles the delay time of each cell. For type 1 and 4 cells the double inversion problem is solved by modifying the functions to fit the complemented outputs. However, for type 2 and 3 cells this solution does not work, since these cells drive cells of the same type. Instead, two different subtypes of type 2 cell are defined: type 2a, which produces complemented outputs from its true inputs and type 2b, which produces true outputs from its complemented inputs. Now, cells along the critical paths are chosen to be of types 2a and 2b alternately. For type 3 cells, inverting the left output of  $C[4,4]$  and  $C[4,10]$  (that drive other type 3 cells) is unavoidable. Inverters must also be added to few type 1 and 2 cells in order to adjust their outputs to the driven cells. For these cells, only the outputs that drive the cells in the same column are inverted again, while the outputs that drive cells to the left remain unchanged. Since critical paths have already been taken care of, the adder speed does not degrade by these inverters. In Figure 6, cells that contain additional inverters have a bar added above their type number.



### Conclusions

We have described the basic ideas behind a gate-to-silicon compiler by walking through a simple and well-known example. The compiler consists of four modules, each of which performs one step of the translation toward silicon level. The first translation is a crude approximation of the final layout, and therefore one or more iterations are needed to achieve a "near optimal" solution.

The novel approach in our compiler is based on (a) the set of synthesis procedures for decomposition of gate macros into small atomic cells and for optimization of obtained cellular structures with respect to environmental and technological parameters, and (b) the set of algorithms for automatic layout of different cell models obtained through decomposition of gate macros.

References

- [BiGa80] Bilgory, A. and Gajski, D. D., "Automatic Cell Generation for Recurrence Structures" University of Illinois at Urbana-Champaign, Department of Computer Science, Report UIUCDCS-R-80-1040, November 1980.
- [Joha79] Johannsen, D., "Bristle Blocks: A Silicon Compiler," Proc. 16th Design Automation Conf., pp 310-313, 1979.
- [MeCo80] Mead, C. A., Conway, L. A., Introduction to VLSI Systems, Addison-Wesley, 1980.
- [SOHT80] Shirakawa, I., Okuda, N., Harada, T., Tani, S. and Ozaki, H., "A Layout System for the Random Logic Portion of MOS LSI," Proc. 17th Design Automation Conf., pp 92-99, 1980.
- [Verg80] Vergnieres, B., "Macro Generation Algorithms for LSI Custom Chip Design," IBM J. Res. Develop., Vol. 24, pp 612-621, 1980.

# SLIM: A Language for Microcode Description and Simulation in VLSI<sup>1</sup>

John Hennessy  
Computer Systems Laboratory  
Stanford University

## Abstract

SLIM (Stanford Language for Implementing Microcode) is a programming language based system for specifying and simulating microcode in a VLSI chip. The language is oriented towards PLA implementations of microcoded machines using either a microprogram counter or a finite state machine. The system supports simulation of the microcode and will drive a PLA layout program to automatically create the PLA.

## 1 Introduction

VLSI chip design has rapidly become an area of great importance and interest. Mead and Conway [6] have proposed a design methodology for VLSI systems that has been widely employed. Their design methodology proposes a chip organization using: finite state control implemented with a PLA, functional units controlled by the PLA, and a set of data paths. This design methodology has been used in a number of large chip designs [5, 1, 2]. The finite state control can be thought of as microcode. Within a design that follows the microcoded control approach, designing and debugging the microcode appears to constitute a significant portion of the work involved in the design process [1, 2]. This paper describes a language for synthesizing the control units of a chip from a high level language description.

Presently few tools exist to assist the user in designing and debugging the microcoded control. Programs to construct PLA's from boolean equations are widespread; however, the difficult component of control unit design is to specify and debug the microcode. This difficulty arises in generating the boolean equations that describe the finite state machine control. Transforming these

---

<sup>1</sup>This research was partially supported by the Joint Services Electronics Program under contract # DAAG29-79-C-0047 and the Defense Advanced Research Projects Agency under contract # NDA903-79-C-0680.

to a PLA layout is tedious and error-prone but mechanically straightforward. Some work has been done on describing PLA's at a higher level [7] and on synthesizing PLA descriptions from low level state machine descriptions in DDL [4].

SLIM (Stanford Language for Implementing Microcode) is a programming language useful for the design of a microcoded system that will employ PLA implementation techniques. Unlike earlier work SLIM is functionally oriented. Control in SLIM is based on a finite state machine, but SLIM deals with objects that can be more abstract than the actual PLA inputs and outputs. The SLIM system supports both microcode simulation and automatic synthesis of the microcoded control function either in ROM or PLA. SLIM will also accommodate either finite state machine control or control with a program counter.

Correct microprograms are both tedious and difficult to write for several reasons. First, the programming language is extremely low level. Typically, the designer must deal with a primitive finite state machine without the benefit of a human-engineered interface. Secondly, many of the microprograms are large. This leads to a relatively complex program without a great deal of structure; this is especially true if the finite state machine is coded as boolean equations. A boolean equation approach makes it difficult to consider altering the microcode, even during the debugging process.

Another major difficulty is the significant level of detail that must be expressed. This leads to one of two pitfalls: either the microcode description is very low level and cluttered with details, which makes it impossible to understand; or the designer uses an ad hoc higher level description of the microcode. An ad hoc description is unsuitable because the translation to the low level microcode must be done by hand, and the description tends to be too informal and vague. Without a higher level standard representation, microcode programs are difficult to write correctly and virtually impossible to understand. The SLIM system is also able to translate the boolean equation representation of the PLA into a layout.

We can summarize the goals of SLIM as

- a symbolic higher level language suitable for designing and documenting the microprogram and oriented towards implementation with PLA technology,
- simulation tools to debug the microcode,
- automatic layout of the PLA based on the microcode.

The SLIM design goals spawn a set of language and system requirements. The microcode simulation requirement implies the ability to describe the subsystems that interact with the microcontroller; we will refer to these subsystems as the *environment*. Describing the environment can be easily done in a conventional programming language, if the interaction with the microcode occurs in a restricted and well defined manner. Separating the micromachine description from the environment description has two benefits. The separation increases comprehensibility of the micromachine structure. A specialized language is also more appropriate for the microprogram design; without the separation the translation process is difficult or impossible.

The environment of the finite state micromachine can be described in a conventional programming language. The environment consists of data structures and variables which can be used to simulate the structure of the subsystems. The environment/controller interface is based on a set of functions and procedures. The functions, which must be type boolean, correspond to the inputs to the microcode machine, while the procedures correspond to outputs. We have chosen Pascal to represent the environment. The Pascal data structures provide additional support in describing the functional components. The wide variety of data types coupled with strong type checking also provides support for checking the microcode and making design restrictions explicit in the SLIM program.

Since the end product of SLIM program is a finite state machine implemented with PLA techniques, a SLIM program must incorporate details about the implementation. This specification should include: mappings between functions and procedures in the environment, actual PLA inputs and outputs, and timing specifications that force outputs to occur earlier or later than they occur in the program. Including these details separately allows a more functional orientation in the microcode description. Lastly, details concerning the actual PLA layout are needed, e.g. the number of PLA's and the positioning on the PLA of each signal.

## 2 Specifying the microcode

A SLIM program consists of a finite state machine. Each state in the SLIM program contains a series of conditional actions that may cause one or more outputs to be high, or may specify the next state. The next state may be specified by default or explicitly. The outputs associated with a given state are conditional on a set of product terms only. Although arbitrary boolean expressions could be used, SLIM does not because it requires a significant amount of processing to transform the expressions to a PLA oriented sum of products form. In this process the number of product terms

added to the PLA may be substantial (up to  $2^n$  terms for an expression of length  $n$ ). The property that – the number of product terms in the PLA is approximately equal to the number of preconditions for the outputs in a SLIM program – has been useful in estimating the PLA size.

There are two major schemes for implementing the state component of a finite state machine. A standard finite state implementation uses a fixed state assignment and includes an encoding of the next-state function in the PLA. An alternative implementation uses a microprogram counter that is incremented under external control. Each approach has benefits that depend on the the microprogram being implemented. The tradeoffs and the advantages of the two different VLSI control implementations are discussed in [3]. SLIM supports both control implementations, provides default next states for program counter implementations, and will support subroutines with call and return in either case.

A SLIM microprogram consists of a set of states listed sequentially. Each state may optionally have a label, which denotes the state name. The specification of the first state is preceded by a set of specifications for outputs which are state independent. Figure 1 shows the format of a the state machine specification.

```
fsm
state-specification (for state independent outputs)
state-name (optional) : [state-specification]
.
.
state-name (optional) : [state-specification].
```

Figure 1: Specifying the state machine

A state specification is a list whose elements are either unconditional actions or conditional commands. A conditional command consists of a condition and a list of actions. A condition consists of a list of one or more product terms that are joined with **or**, and a product term is a series of predicates joined with **and**. A predicate must be a call to a function in the environment; predicates correspond to one or more PLA inputs. The interpretation of the command is: if the entire condition evaluates to true, then the actions should be executed. If there are no predicates, the condition is assumed to be true and the action is always executed in that state. The form of a state specification is given in Figure 2.

```
if  $p_1$  and ..... and  $p_n$  or  $q_1$ ..... or  $q_m$  => action
```

Where the  $p_i$  are function invocations and the  $q_j$  are product terms,  
like the first term.

Figure 2: State specifications

Each state may contain a list of such specifications and the entire state is bracketed. During simulation, state specifications are evaluated and executed sequentially, but in the actual PLA implementation these operations will occur in parallel. Therefore, side effects between procedures that are outputs and functions that are inputs in the same state should be employed with great care.

## 2.1 Actions

There are two types of actions allowed: outputs and state change operations. A list of actions can be used as a single compound action by bracketing the list. Outputs are invocations of procedures in the environment and correspond to PLA outputs. The state change directives dictate the next state. All state change directives have effect only after the current state is completed; thus, all state specifications with true conditions will be executed in a state. The state change directives are:

**next** *state-name* - makes *state-name* the next state.

**call** *state-name* - does a microcode call to the routine at *state-name*.

**return** - returns to the state sequentially following the calling state.

## 2.2 A short example

Figure 3 shows the finite state machine controller for the traffic light example from [6]. (The entire example is given in the appendix.) The state independent component is for simulation purposes. The procedures *Farmlight* and *Highlight* alter the color (which is a parameter) of the traffic light at the farmroad and the highway. *Timeout* looks for the timeout condition, which is either short or long as dictated by the parameter. The function *Cars* corresponds to the test for a car.

**Figure 3:** Microcode specification for the Mead/Conway traffic controller

```
fsm
[ getinput; timer ] { state independent component }
highgrn: [ highlight(green); farmlight(red); {Highway green and farmroad red}
          if notcars or nottimeout(long) => next highgrn;
          if cars and timeout(long) => [ starttimer; next highyel ] ]
highyel: [ highlight(yellow); farmlight(red); {Highway yellow and farmroad red}
          if nottimeout(short) => next highyel;
          if timeout(short) => [ starttimer; next farmgrn ] ]
farmgrn: [ highlight(red); farmlight(green); {Highway red and farmroad green}
          if cars and nottimeout(long) => next farmgrn;
          if notcars or timeout(long) => [ starttimer; next farmyel ] ]
farmyel: [ highlight(red); farmlight(yellow); {Highway red and farmroad yellow}
          if nottimeout(short) => next farmyel;
          if timeout(short) => [ starttimer; next highgrn ] ].
```

### 3 Defining the Relationship to the PLA

The relationship between the microcode specification of the control program and the PLA is defined by: declaring the input and output signals for the PLA and defining the mappings between environment functions/procedures and input/output signals. The expressive power of this mapping is one of the advantages of SLIM.

#### 3.1 Defining input and output signals

PLA signals are defined by means of input and output signal declarations, which appear just before the definition of the environment procedures. Signal declarations begin with the keyword **inputs** or **outputs**, as appropriate. The general form of each declaration is then:

{name [ '(' bounds ')' ] } [ ':' parameters ] ';' ;

The list of names are the names of input or output signals being declared. The optional bounds designator indicates whether a particular signal is a single bit or a vector of bits. In the latter case the line can be treated as an integer-encoded number; the order of the bounds (low to high or high to low) specifies the order of the lines in the signal vector. If any optional parameters appear they are associated with all input/output names in the declaration. Table 1 defines the legal parameters.

Table 1: Signal parameters

{Syntax	Meaning	For input/output}
<b>pla</b> ( <i>n</i> )	Associate signal with pla # <i>n</i>	both
<b>top</b>	Position signal on top of pla	both
<b>bottom</b>	Position signal on bottom of pla	both
<b>renames</b> ( <i>id</i> )	Give the signal <i>id</i> another name	both
<b>earlier</b> ( <i>n</i> )	Move the signal <i>n</i> states earlier	output
<b>later</b> ( <i>n</i> )	Move signal <i>n</i> states later	output

A signal declaration specifies physical placement information using the directives **top** and **bottom**. The order of the signals on the PLA is given by the order of their declaration. The state signals are added by SLIM and appear last in the PLA inputs and first in the outputs; this facilitates interconnection. When more than a single PLA is specified SLIM determines which outputs should appear from which PLA's (by declaration or default to PLA 1). Only the necessary inputs are generated for each PLA; these are based on the outputs that are specified in that PLA.

The optional pipelined directives, i.e. **earlier** and **later**, move an output signal forward or backward in the state graph. This is very useful when a particular signal, which is logically associated



with a single operation, must occur earlier. A frequently occurring example of this is precharging or enabling of alu's. Although the functional operation *add* appears to occur in a single state the alu must be precharged/enabled one state earlier. The pipelined directives provide a convenient way to express such relationships without adding needless details to the microcode description. If an output signal *x* appears in a state *s* conditional on input *c* and *x* is pipelined *earlier(i)*, then the output *x* will appear, conditional on *c*, in all the states that precede *s* by *i* states. Although pipelining can be done into both predecessor and successor states, by far the most common situation is pipelining into the immediate successor state. SLIM finds all predecessor or successor states, including those that occur when the state that is pipelined from is the target of a branch or call. Pipelining is not permitted across a procedure return, i.e. in the state following a call. The **renames** directive gives a signal another name, without associating the other characteristics (e.g. pipelining) of the renamed signal. This is useful if a particular signal must be pipelined nearly all the time, but occasionally nonpipelined generation of the signal is needed.

### 3.2 Describing the relationship between environment and outputs

Since a procedure or function in the environment can logically correspond to one or more signals, SLIM provides a method of defining the mapping between environment routines and signals. This method allows the microcode description to be functionally oriented, and to significantly decrease the amount of code needed to describe the PLA implementation of the microcode.

The mapping between environment procedures and signals to be generated in the PLA is given in the definition section of an environment procedure or function. The definition section starts with the keyword **definition** and appears immediately after the function or procedure header. Procedures in the environment without a definition section are presumed to be for simulation purposes only. The definition section consists of a list of signal definitions which are separated by semicolons; the definition section is terminated by **end**.

A signal definition has the form:

[ *pattern-string* : ] *signal-expression*

The optional *pattern-string* is used to specify different signal combinations based on the values of the parameters to the environment procedure. The *pattern-string* consists of a list of string patterns separated by commas and enclosed in parenthesis. If the pattern list matches the list of actual parameters in a call to this procedure, then the signals in the signal list are generated as outputs. Each string pattern can either be a alphanumeric string or a "\*\*\*". The latter is a wild card match, indicating that any actual parameter value should generate a match for the corresponding parameter.

The *signal-expression* specifies what signals to generate; it may also contain invocation of other environment procedures. Before it is evaluated any identifiers in the signal-list that correspond to formal parameters are replaced by the actual parameter values in the call for which signals are being generated. The types of signal expressions are defined in Table 1.

<i>Signal-expression</i>	<i>Meaning</i>
signal name	emit the signal
procedure-name(parameters)	emit the signals for the named procedure
<i>signal-expression</i> and <i>signal-expression</i>	emit both sets of <i>signal-expressions</i>
<i>expr</i> <sub>1</sub> & <i>expr</i> <sub>2</sub>	Emit <i>expr</i> <sub>2</sub> concatenated to <i>expr</i> <sub>1</sub>
signal name = integer constant	emit encoded constant to the signal vector
not <i>signal-expression</i>	emit inverse of a simple <i>signal-expression</i>
<i>signal-name</i> [constant]	emit a single signal within a signal vector

Table 2: *Signal-expressions*

If the signal identifier is an environment procedure and not an signal name, the definition section of the referenced environment procedure is used for that signal. Naturally, the procedure name can be followed by parameter strings. This facility allows multi-level environment procedures to produce signals by composing the definition list in each procedure.

In Figure 4 some input/output declarations and two of the procedures from the Mead/Conway traffic light example are given. The highway traffic light is encoded as a two-element vector; the input testing for cars is a single bit. Note that PLA signals may have the same name as components of the Pascal program.

Figure 4: An example from the Mead/Conway Traffic Controller

```

type      colortype = (green,yellow,red);

inputs    c: bottom;
outputs   h1[1..0] : bottom;

procedure highlight(color: colortype);
definition
    (green): h1 = 0 ;
    (yellow): h1 = 1 ;
    (red): h1 = 2 ;
begin     h1 := color end;

function cars :boolean;
definition c;
begin     cars := (c = 1) end;

```

## 4 Using SLIM

A SLIM program can be used to drive a microcode simulation as well as generate a PLA layout. A SLIM simulation requires a microcode description with all of the environment procedures and functions. The simulation is presently done by creating a Pascal program which embodies the semantics of the microcode. A SLIM simulation can be requested with state tracing.

PLA generation is a straightforward process, which is done in two parts. The first part analyzes the microcode structure and creates product term lists for each output. The effect of signal definition and pipelining is integrated before making these lists. The PLA layout is then done by a separate program which inputs the signal descriptions and the product term lists. The intermediate form uses boolean expressions; this allows the use of any PLA generator that accepts boolean equations as input and the use of PLA optimizers prior to layout.

Another program in the SLIM system can be used to assist in choosing a state encoding (applicable only for finite state implementations). The program accepts output from SLIM with the state entries unencoded. It computes a matrix whose  $i,j$  entry is the saving in product term count that will result if states  $i$  and  $j$  are encoded so that they can be uniquely distinguished from all other states with a single product term.

### 4.1 Ensuring microcode correctness

There are several useful types of debugging and checking of microcode that can be done in the process of simulation. Most important among these are detecting potential errors which arise because the simulation does not exactly match the PLA implementation, or because the microcode does not employ the environment in a manner that the hardware is designed to support. Another class of errors may arise because the simulation may fail to test all possible combinations of inputs or fail to test all states.

The major reasons that the simulation and PLA implementation might behave differently is because the simulation treats outputs, environment procedures, and the state as unique entities in a sequential manner. In the PLA these objects are interrelated. Problems such as assigning two next states are resolved into a single, well defined action in the simulation, but these actions result in a disaster in the PLA implementation, since both sets of state bits are set high. Certain classes of these errors can be caught by predefined, microcode independent methods, but others require a more general scheme, which we can also employ to find errors concerning the use of the hardware environment by the

microcode. SLIM checks for common sorts of errors, such as failing to assign a next-state in a finite state machine implementation, or attempting to assign more than one next-state.

Many of the hardware/microcode inconsistencies arise from situations where certain outputs are being incorrectly used, perhaps with respect to timing, or the hardware is being instructed to preform some task it is not physically able to undertake. Many of the latter types of errors can be caught using a strictly type-checked environment specification. For example, suppose that the register file on some microcoded processor is divided into two sections in such a way that two registers from the same section can not be gated to the alu (many hardware micromachines have this property). Microcode errors that arise because two registers from the same section are being sent to the alu can be detected by defining the machine structure with two different types for the registers and specifying that the alu environment procedures have two parameters — one from each register section. This class of simple errors is detected at compile-time.

A more complex class of errors can not be detected with a straightforward compile-time scheme. Some examples of this type of error are: attempts to use the bus for two different quantities in the same time frame, overlapping use of environment hardware (such as an alu), and incorrect timing of an output in a state. Many of these errors can be detected during simulation using a set of assertions, which can be checked during simulation. We divide these assertions into two groups: invariant assertions and state dependent assertions. The invariant assertions specify conditions which must hold regardless of the current state, e.g. if an alu output occurs in this state, the alu was precharged in the previous state and was not doing any other operation. State dependent assertions specify properties which should hold at a particular state, e.g. a certain part of the machine should have a certain value.

In SLIM anywhere an action can occur, an assertion can be specified. Although the assertion generates code for simulation purposes, no PLA entries are affected or generated. Assertions are only used to ensure that certain properties hold. An assertion has the form `assert` invocation, where invocation must be the invocation of a boolean function. Whenever execution reaches an `assert` statement at simulation time, the simulation invokes the specified function. If the function returns false the simulation is halted with an appropriate error message.

In using SLIM, we have found that the expressive power of SLIM's pipelining and signal definitions is one of its major advantages. However, the mechanism can also lead to errors, since the specifications are not reflected in the simulation. To assist in ensuring that the signal specifications in

a SLIM program are consistent and correct, two types of output-generation checking are supported. Pipeline checking will cause a warning to be generated whenever a signal component both occurs in a state and is pipelined into that state from another state. This appears to catch most errors in the use of pipelining. Another powerful check is examining sets of mutually exclusive signals. A SLIM program can specify one or more *exclusive sets*. SLIM will check that no two signals in the same exclusive set can be generated in the same state.

## 5 Current status and concluding remarks

This paper describes SLIM, a language and processing system for describing microcode whose implementation orientation is PLA based. The purposes of this language are: to document the microcode at a reasonable, logical level while providing a firm specification; to allow extensive simulation, debugging, and error detection; and to automatically create the PLA layout necessary to implement the microcode description.

SLIM has been working for approximately one year. It is coded in standard Pascal. To date, experience with SLIM has been highly favorable. It has been used in the development of two large chip designs [1, 2], both of these contain extensive microcoding. It has also been used in a number of smaller projects with favorable results.

The most significant observation we have made in using SLIM is the enormous significance of the control function and its design. For large projects, we have found that 60-75% of the design time is spent in constructing and debugging the control as specified by SLIM. A large amount of this time is spent in constructing an accurate functional specification of the data components as a SLIM environment. In many instances, the construction of SLIM environment has uncovered bugs in the data components being described. The specification of the control program itself is also time consuming especially in the debugging process.

There are many interesting questions concerning the applicability of SLIM that have not been investigated. It would be interesting to examine the use of SLIM for microcode machines whose architecture is not strictly PLA based, but whose microcontrol is straightforward. We are also interested in supporting a wide variety of PLA implementations and in PLA optimization.

## Appendix 1. Annotated Syntax of SLIM

This is the syntax for the non-Pascal portion of SLIM. Nonterminal symbols appear to the left of =; terminal symbols in the grammar are distinguished by being in quotes. The metasyntax  $[\alpha]$  means that the string  $\alpha$  is optional, and  $\{\alpha\}$  means that the string  $\alpha$  may be repeated zero or more times. Comments can appear at the end of a production and are started with --.

```

Program = 'program' <id> 'Programparms ';' Outerblock
Outerblock = Constpart Typedefpart Vardeclpart Iopart Procpart Fsm -- A Pascal program with a fsm body
Procheading = 'procedure' <id> 'Formalparms ';' Definitionpart -- Procedures contain definitions
Funcheading = 'function' <id> 'Formalparms ';' <id> ';' Definitionpart
Iopart = ['inputs' Spec { ';' Spec }] ['outputs' Spec { ';' Spec}] -- Input/output declarations
Spec = Vector { ';' Vector } [';' Parameter {Parameter} ';' ] -- An input/output vector
Vector = <id> [ '[' <int> ';' <int> ']' ] -- Vector has integer bounds
Parameter = 'pla' '(' <int> ')' -- PLA number
    = 'top' -- Top of PLA
    = 'bottom' -- Bottom of PLA
    = 'earlier' '(' <int> ')' -- Pipeline into earlier states
    = 'later' '(' <int> ')' -- Pipeline into later states
    = 'renames' '(' <id> ')' -- Rename a signal (without pipelining)
Definitionpart = [ 'definition' Definition {Definition} ]
Definition = [ '(' Patternlist ')' ';' ] Output { 'and' Output } ';' -- Definition is a series of pattern lists
Patternlist = Pattern { ';' Pattern } -- Each pattern list must match the parameter list
Pattern = '*' -- Wild card match
    = <id> -- Name match
Output = [ 'not' ] Plainoutput -- Outputs can be inverted
Plainoutput = Invocation [ '&' Output ] -- Outputs can be composed by concatenation
    = <id> '=' Constant -- A vector can output an encoded integer
    = <id> '[' <int> ']' -- A single line from a vector can be made high
Fsm = 'fsm' Stateindpart {State} ';' -- The FSM contains a state independent part and a list of states
Stateindpart = [ 'Statespecifiers ';' ]
State = [ <id> ';' ] [ 'Statespecifiers ';' ] -- States are optionally labelled
Statespecifiers = Statespec { ';' Statespec }
Statespec = [ 'if' Cond { 'or' Cond } '=' > Action ] -- A state is conditional on a sum of product terms
Cond = { Invocation 'and' } Invocation -- Form of a product term, the invocations are functions
Invocation = <id> [ '(' Constant { ';' Constant } ')' ] -- Limited
function invocation, constant can be a variable
Action = [ ']' Action { ';' Action } ']' -- Composite action
    = 'assert' invocation -- Assert action
    = Invocation -- Procedure invocation
    = 'next' <id> -- Goto specified State
    = 'call' <id> -- A microcode subroutine call
    = 'return' -- A microcode subroutine return

```

## Appendix 2. More Examples

### The Full Traffic Controller from Mead/Conway

```

program traffic(input,output);
const  short = 2;      long  = 4;
type   colortype = (green,yellow,red);
       signaltype = 0..1;
var     time: integer;   hl,fl: colortype;      c: signaltype;
inputs  c,tl,ts : bottom;
outputs st,hl[1..0],fl[1..0] :bottom;
procedure getinput; { for simulation purposes only }
begin    write('cars? ');read(c); end;
procedure timer; { for simulation purposes only }
begin    if time < long then time := time + 1 end;
procedure highlight(color: colortype);
definition
    (green): hl = 0 ;
    (yellow): hl = 1 ;
    (red): hl = 2 ;
begin    hl := color end;
procedure farmlight(color: colortype);
definition
    (green): fl = 0 ;
    (yellow): fl = 1 ;
    (red): fl = 2 ;
begin    fl := color end;
procedure starttimer;
definition st;
begin    time := 0 end;
function cars :boolean;
definition c;
begin    cars := (c = 1) end;
function notcars :boolean;
definition not c;
begin    notcars := not cars end;
function timeout(length: integer) :boolean;
definition
    (long): tl ;
    (short): ts ;
begin    timeout := (time >= length) end;
function nottimeout(length: integer) :boolean;
definition
    (long): not tl ;
    (short): not ts ;
begin    nottimeout := not timeout(length)end;

fsm
    [ getinput; timer ] { state independent component }
highgrn: [ highlight(green); farmlight(red);
          if notcars or nottimeout(long) => next highgrn;
          if cars and timeout(long) => [ starttimer; next highyel ] ]
highyel: [ highlight(yellow); farmlight(red);
          if nottimeout(short) => next highyel;
          if timeout(short) => [ starttimer; next farmgrn ] ]
farmgrn: [ highlight(red); farmlight(green);
          if cars and nottimeout(long) => next farmgrn;
          if notcars or timeout(long) => [ starttimer; next farmyel ] ]
farmyel: [ highlight(red); farmlight(yellow);
          if nottimeout(short) => next farmyel;
          if timeout(short) => [ starttimer; next highgrn ] ].

```

### Example — Computing GCD

```

program test (input,output);
var x,y: integer;
inputs
    eq1,eq0,gtx, gty: bottom;
outputs
    aluop[1..2] : bottom ;
    enablex,enabley: top earlier (1);
procedure init;
begin      read(x); read(y);      end;
procedure subt (var a,b: integer);
definition
    enable & a and enable & b and aluop = 1;
begin a := a-b end;
function greater (x,y:integer): boolean;
definition
    gt & x ;
begin greater := x>y end;
function equal (x,y:integer): boolean;
definition eq & y;
begin eq := x = y; end;
function ne(x,y:integer): boolean;
definition not equal (x,y);
begin ne := not equal(x,y); end;

fsm
[;]
one : [ init ;
      assert ne(y,0);
      if equal(x,0) => next endstate ]
      [ call two ]
      [ next one ]
two:  [ if greater(x,y) => [subt (x,y); next two ];
      if greater(y,x) => [subt (y,x); next two ]]
three: [ assert equal(x,y);
      if equal(x,1) => [writeln(1); return];
      if ne(x,1) => [writeln(y); return]]
endstate: [ halt ] .

```



## References

1. Clark, J.H. "A VLSI Geometry Processor for Graphics." *Computer* 13, 7 (July 1980), 59-68.
2. Clark, J.H. and Hannah, M.R. "Distributed Processing in a High-Performance Smart Image Memory." *Lambda* 1, 3 (1980), 40-45.
3. Clark, J.H., Hennessy, J.L., Hannah M.R. A comparasion of two different VLSI control structures. Computer Systems Laboratory, Stanford University, Dec, 1980.
4. Duley, J.R. and Dietmeyer, D.L. "Translation of DDL digital system specification to Boolean equations." *IEEE Trans. Computers* c-18, 4 (Apr 1969), 305-313.
5. Holloway J., Steele, G., Sussman, G., Bell, A. The Scheme-79 Chip. Tech. Rept. 599, Artificial Intelligence Laboratory, MIT, Jan, 1980.
6. Mead, C. and Conway, L.. *Introduction to VLSI Systems*. Addison-Wesley, Menlo Park, Ca., 1980.
7. Weber, H. High Level Design for Programmed Logic Arrays. Proceedings of Fourth Conf. on Computer Hardware Description Languages, May, 1979, pp. 96-101.



## Signal Delay in RC Tree Networks\*

Paul Penfield, Jr.\*\*

Jorge Rubinstein\*\*\*

## ABSTRACT

In MOS integrated circuits, signals may propagate between stages with fanout. The exact calculation of signal delay through such networks is difficult. However, upper and lower bounds for delay that are computationally simple are presented in this paper. The results can be used (1) to bound the delay, given the signal threshold; or (2) to bound the signal voltage, given a delay time; or (3) to certify that a circuit is "fast enough", given both the maximum delay and the voltage threshold.

## I. Introduction

In MOS integrated circuits, a given inverter or logic node may drive several gates, some of them through long wires whose distributed resistance and capacitance may not be negligible. There does not seem to be reported in the literature any simple method for estimating signal propagation delay in such circuits, nor is there any general theory of the properties of RC trees, as distinct from RC lines. This paper presents a computationally simple technique for finding upper and lower bounds for the delay. The technique is of importance for VLSI designs in which the delay introduced by the interconnections may be comparable to or longer than active-device delay. This can be the case for wiring lengths as short as 1 mm, with 4-micron minimum feature size. The importance of this technique grows as the wiring lengths increase or the feature size decreases.

---

\*This work was supported in part by Digital Equipment Corporation, in part by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under Contract N00014-C-80-0622, and in part by the Air Force under Contract Number AFOSR 4-9620-80-0073.

\*\*Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Room 36-575, Cambridge, MA 02139; telephone (617) 253-2506.

\*\*\*Digital Equipment Corporation, 75 Reed Road, Hudson, MA 01749; telephone (617) 568-4835.

Consider the circuit of Figure 1. The slowest transition (and therefore presumably the one of most interest) occurs when the driving inverter shuts off and its output voltage rises from a small value to  $V_{DD}$ . During this process the various parasitic capacitances on the output are charged through the pullup transistor. Figure 2 shows a simple model of this circuit for timing analysis. The pullup, which is nonlinear, is approximated by a linear resistor, and the transition is represented by a voltage source going from 0 to  $V_{DD}$  at time  $t = 0$ . (Later, for simplicity, a unit step will be considered instead.) The polysilicon lines are represented by uniform RC lines. The resistance of the metal line is neglected, but its parasitic capacitance remains. Capacitances associated with the pullup source diffusion, contact cuts, and the gates being driven are included. Any nonlinear capacitances are approximated by linear ones.

In general, the circuit response cannot be calculated in closed form. The results of this paper can be used to calculate upper and lower bounds to the delay that are very tight in the case where most of the resistance is in the pullup. The theory as presented here does not explicitly deal with nonlinearities and therefore does not apply to signal propagation through pass transistors. A more complete discussion of this theory will appear elsewhere [1].

## II. Statement of the Problem

An RC tree is defined as follows. Consider any resistor tree with no node at ground. From each node in this tree a capacitor to ground may be added, and any resistor may be replaced by a distributed RC line. Although nonuniform RC lines may appear in an RC tree, for simplicity, the examples in this paper involve only lumped resistors and capacitors and uniform RC lines. An RC tree has one input and any number of outputs. Side branches may or may not end in a node that is considered as an output; in fact, outputs may be taken anywhere in the tree. Nonuniform RC lines are special cases of RC trees, without any side branches. An important property of RC trees is that there is a unique path from any point in the tree to the input.

The tree representing the signal path is driven at the input with a unit step voltage. Gradually the voltages at all other nodes, and in particular at all the outputs, rise from 0 to 1 volt. It is assumed that the output voltages cannot be calculated easily. The problem is to find simple upper and lower bounds for the output voltages, or, equivalently, to find upper and lower bounds for the delay associated with each output.

## III. Analytical Theory

Consider any output node  $e$ , and any lumped capacitor at node  $k$  with

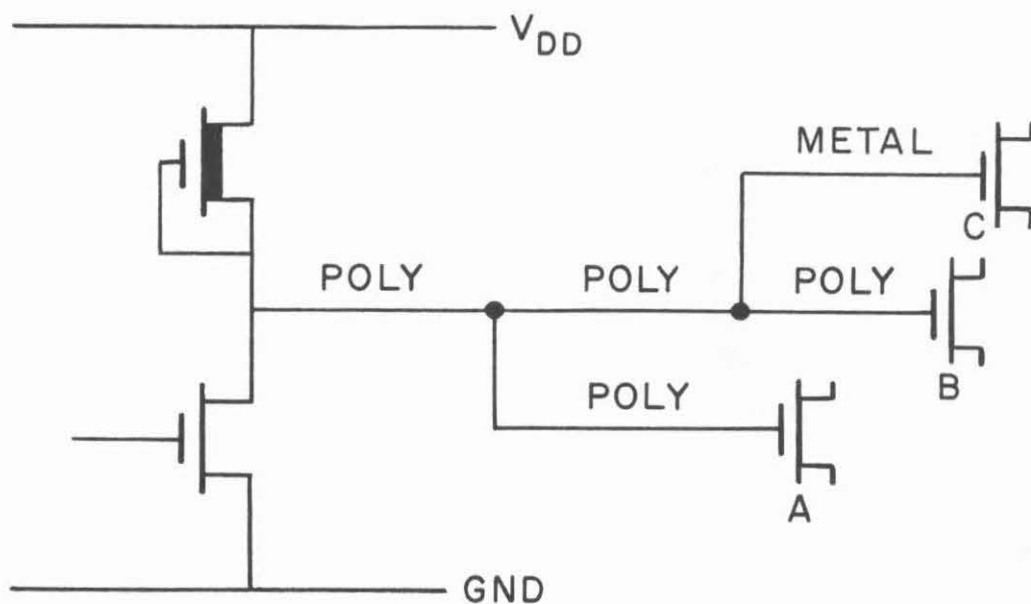


Figure 1. Typical MOS signal-distribution network. The inverter is shown driving three gates.

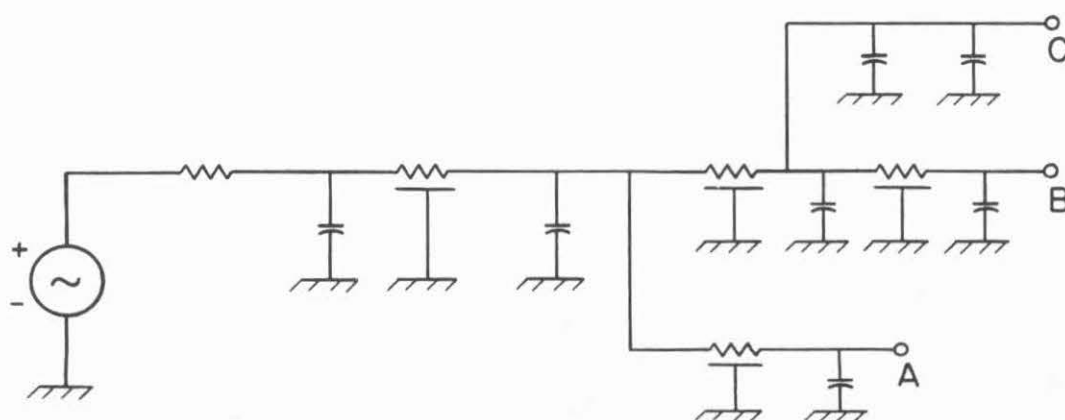


Figure 2. Linear-circuit model for the network of Figure 1. The voltage source is a step at time  $t = 0$ .

capacitance  $C_k$ . For the moment consider only lumped capacitors; the theory is similar if the distributed lines are considered also. One may think of many-stage approximations for the distributed lines, or one may convert some summations in the formulas below to a form including both summations over lumped capacitors and integrals over distributed ones.

The resistance  $R_{ke}$  is defined as the resistance of the portion of the (unique) path between the input and  $e$ , that is common with the (unique) path between the input and node  $k$ . In particular,  $R_{ee}$  is the resistance between input and output  $e$ , and  $R_{kk}$  is the resistance between the input and node  $k$ . Thus  $R_{ke} \leq R_{kk}$  and  $R_{ke} \leq R_{ee}$ . For an example, see Figure 3.

The sum (over all the capacitors in the network)

$$T_{De} = \sum_k R_{ke} C_k \quad (1)$$

has the dimensions of time and is in general different for each output. It is equal to the first-order moment of the impulse response, which has been called "delay" by Elmore [2]. This constant alone can be used to generate a lower bound to the step response. Two other time constants and a set of tighter upper and lower bounds using them will be given at the end of this section.

Let  $v_k(t)$  and  $v_e(t)$  be the voltages at the node  $k$  and output  $e$  respectively, in response to a unit step excitation. The current  $C_k dv_k/dt$  that feeds the capacitor  $C_k$  contributes to the voltage drop between the input and the output  $e$  by the amount  $R_{ke} C_k dv_k/dt$  as it flows through the resistance  $R_{ke}$ . The net voltage drop  $1 - v_e(t)$  is obtained by adding the

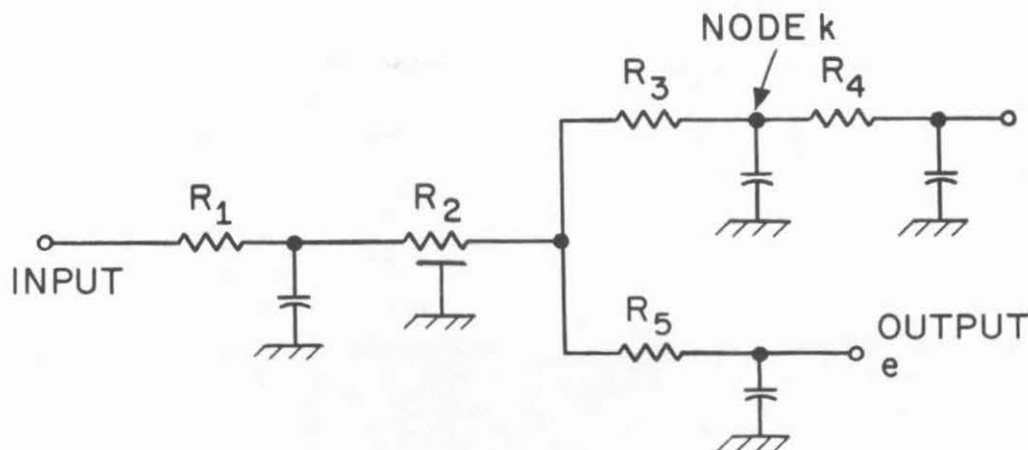


Figure 3. Illustration of resistance terms. For this network,  $R_{ke} = R_1 + R_2$ ,  $R_{kk} = R_1 + R_2 + R_3$ , and  $R_{ee} = R_1 + R_2 + R_5$ .

contributions from all the currents in the tree:

$$1 - v_e(t) = \sum_k R_{ke} C_k \frac{dv_k}{dt} . \quad (2)$$

Integration of the right-hand side of (2) from 0 to  $\infty$  yields  $T_{De}$ , since the voltages are assumed to rise from 0 at  $t = 0$  to 1 when  $t$  approaches  $\infty$ , everywhere in the tree. Thus  $T_{De}$  is equal to the area above the unit step response  $v_e(t)$  but below 1, as indicated in Figure 4. Since  $v_e(t)$  increases monotonically (a fact proven elsewhere [1]), no rectangle with one corner on  $v_e(t)$  and bounded by the lines  $t = 0$  and  $v_e(t) = 1$  can have an area greater than  $T_{De}$  (see Figure 4), i.e.,

$$t[1 - v_e(t)] \leq T_{De} . \quad (3)$$

This expression yields a lower bound for  $v_e(t)$ ,

$$v_e(t) \geq 1 - \frac{T_{De}}{t} . \quad (4)$$

This result illustrates how a suitably defined time constant  $T_{De}$  can be used in a bound for the step response. The computation of  $T_{De}$  and the bound are much simpler than the exact calculation of the response, especially for RC trees with distributed lines.

More complete and tighter bounds require two additional time constants  $T_p$  and  $T_{Re}$  to be defined:

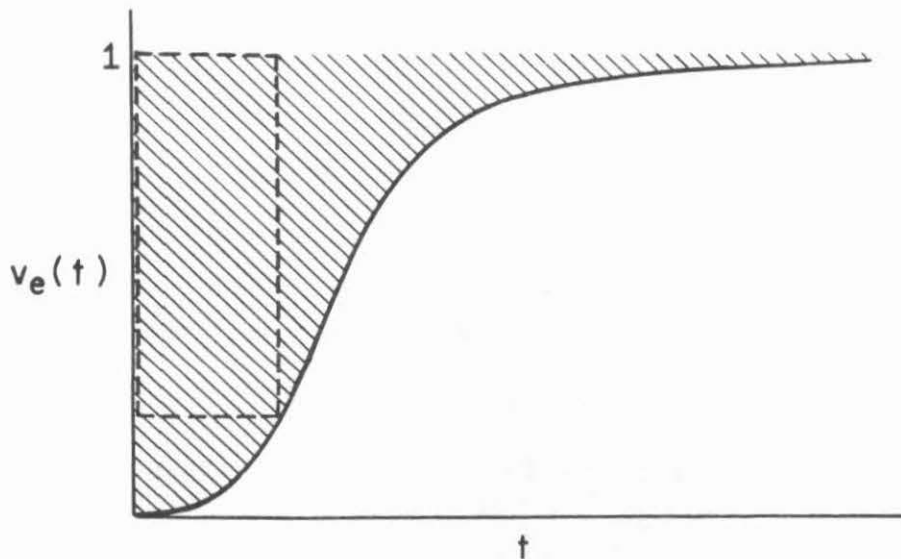


Figure 4. The shaded area is equal to  $T_{De}$  and the rectangle has smaller area.

$$T_P = \sum_k R_{kk} C_k \quad (5)$$

$$T_{Re} = (\sum_k R_{ke}^2 C_k) / R_{ee} \quad (6)$$

Both summations extend over all the capacitors of the network. As with  $T_{De}$ , these additional time constants can be computed easily, even in the presence of distributed lines, and while  $T_{Re}$  is in general different for different output nodes,  $T_P$  is the same for all outputs. It is easily seen that

$$T_{Re} \leq T_{De} \leq T_P \quad (7)$$

For nonuniform RC lines (i.e., RC trees without side branches)  $T_{De} = T_P$ . For a single uniform RC line,  $T_P = T_{De} = RC/2$ , and  $T_{Re} = RC/3$ . Lower bounds, tighter than (4), and upper bounds can both be derived in terms of these three characteristic times. A detailed derivation [1] leads to the upper bounds

$$v_e(t) \leq 1 - \frac{T_{De} - t}{T_P} \quad (8)$$

$$v_e(t) \leq 1 - \frac{T_{De}}{T_P} e^{-t/T_{Re}} \quad (9)$$

and lower bounds

$$v_e(t) \geq 0 \quad (10)$$

$$v_e(t) \geq 1 - \frac{T_{De}}{t + T_{Re}} \quad (11)$$

$$v_e(t) \geq 1 - \frac{T_{De}}{T_P} e^{(T_P - T_{Re})/T_P} e^{-t/T_P} \quad (12)$$

where (12) applies if  $t \geq T_P - T_{Re}$ . The tightest upper bounds are (8) for small  $t$  and (9) for large  $t$ . The tightest lower bounds are (10) for  $t \leq T_{De} - T_{Re}$ , (11) for  $T_{De} - T_{Re} \leq t \leq T_P - T_{Re}$ , and (12) for  $T_P - T_{Re} \leq t$ .

Bounds for the time, given the voltage, are possible because the voltage is a monotonic function of time. Of course

$$t \geq 0 \quad (13)$$

and in addition, (8) and (9) can be inverted to yield



$$t \geq T_{De} - T_P[1 - v_e(t)] \quad (14)$$

$$t \geq T_{Re} \ln \frac{T_{De}}{T_P[1 - v_e(t)]} \quad (15)$$

and (11) and (12) yield

$$t \leq \frac{T_{De}}{1 - v_e(t)} - T_{Re} \quad (16)$$

$$t \leq T_P - T_{Re} + T_P \ln \frac{T_{De}}{T_P[1 - v_e(t)]} \quad (17)$$

where (17) only applies if  $v_e(t) \geq 1 - T_{De}/T_P$ . The general form of all these bounds is illustrated in Figure 5.

These bounds, (8) to (12) for voltage, and (13) to (17) for time, constitute the major result of this paper.

#### IV. Practical Algorithms

One way to use the inequalities of the previous section is to consider the overall RC tree, and compute for each capacitor the appropriate  $R_{ke}$  and

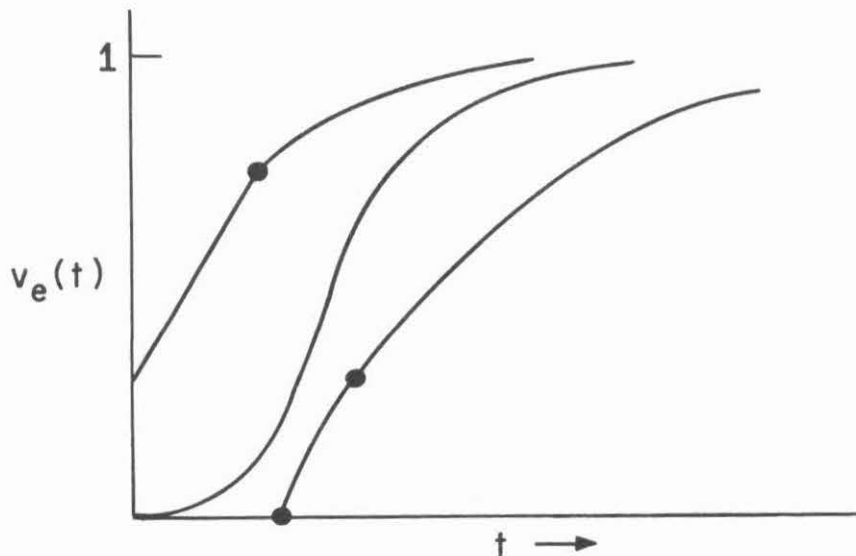


Figure 5. Form of the bounds, with the distances from the exact solution exaggerated for clarity.

$R_{kk}$  so that  $T_p$ ,  $T_{De}$ , and  $T_{Re}$  for each output can be found. Of course for distributed lines the sums are replaced by appropriate integrals. In this approach, the calculations necessary for each output require time proportional to the square of the number of elements.

An alternate approach is to build up the network by construction, and calculate independently for each of the partially constructed networks enough information to permit the final calculation of  $T_p$ ,  $T_{De}$ , and  $T_{Re}$ . A recursive definition of RC trees is given below, and if the network is expressed in these terms rather than in the form of a schematic diagram, the resulting expression can be used as a guide for the calculations. The computation time for each output is proportional to the number of elements, rather than the square of the number. Programs that implement this approach appear below.

For simplicity, the tree is assumed to consist of lumped capacitors, lumped resistors, and uniform (not nonuniform) RC lines. Only one output is considered; a more general set of programs is described elsewhere [1]. Only one primitive element, a uniform line (URC) is necessary. If either the resistance of the line or the capacitance is zero, the line reduces to a lumped capacitor or resistor. The line is denoted URC  $R,C$  where  $R,C$  is a vector of length 2 consisting of the resistance and capacitance of the line, in that order. A capacitor is written URC  $0,C$  and a resistor URC  $R,0$ . Figure 6 shows a way of converting a subtree into a side branch and a way of cascading two subtrees. The topology of any RC tree can be denoted by an expression using only these two functions, WB and WC.

Example: The network shown in Figure 7 is a tree with one side branch and may be denoted

$$\begin{aligned} &(\text{URC } 15 \ 0) \text{ WC } (\text{URC } 0 \ 2) \text{ WC } (\text{WB } (\text{URC } 8 \ 0) \text{ WC } \text{URC } 0 \ 7) \text{ WC } (\text{URC } 3 \ 4) \\ &\text{WC } \text{URC } 0 \ 9. \end{aligned} \quad (18)$$

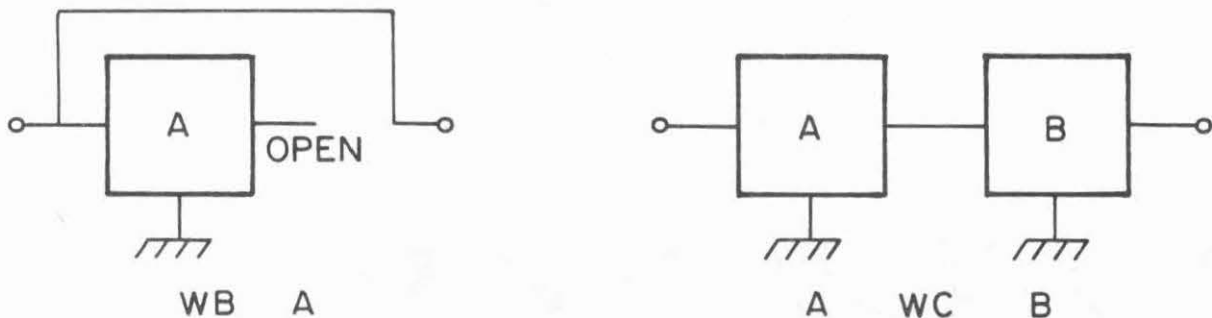


Figure 6. Wiring functions for interconnecting elements or subtrees. Here  $A$  and  $B$  are previously defined RC trees.

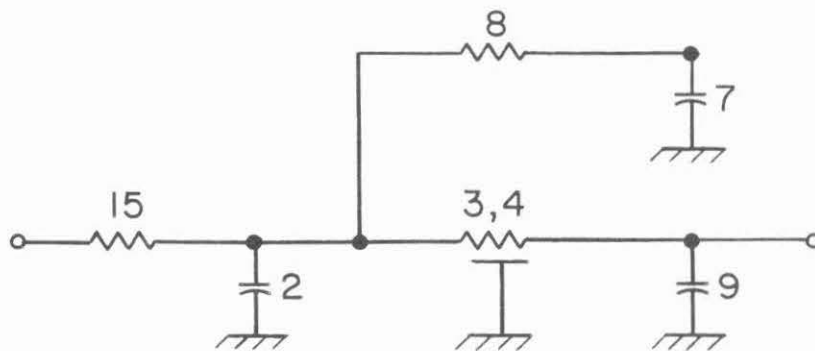


Figure 7. Example network. Parameter values are in ohms and farads.

An expression such as (18) can be used as a guide for the calculations if each function shown corresponds to the calculation of partial results which are sufficient to allow further calculations. The following information is adequate at each stage in the construction of the network:

- (1) Total capacitance  $C_T$ .
- (2)  $T_P$  of the network as constructed so far.
- (3) Considering port 2 as the output,  $R_{22}$ ,  $T_{D2}$ , and  $T_{R2}$ . (For convenience, the product  $R_{22}T_{R2}$  is used in the programs below instead of  $T_{R2}$ .)

Each of the quantities identified above pertains to the particular subnetwork and can be calculated from a knowledge of that subnetwork alone, independent of how the subnetwork may later be wired together with other subnetworks. As an example of the use of these quantities during construction of the network, consider the cascade operation WC. The objective is to find  $C_T$ ,  $T_P$ ,  $R_{22}$ ,  $T_{D2}$ , and  $T_{R2}$ , of the cascade A WC B from the corresponding quantities for its two arguments, A and B. The formulas for calculating these are

$$C_T = C_{TA} + C_{TB} \quad (19)$$

$$T_P = T_{PA} + T_{PB} + R_{22A}C_{TB} \quad (20)$$

$$R_{22} = R_{22A} + R_{22B} \quad (21)$$

$$T_{D2} = T_{D2A} + T_{D2B} + R_{22A}C_{TB} \quad (22)$$

$$T_{R2}R_{22} = T_{R2A}R_{22A} + T_{R2B}R_{22B} + 2R_{22A}T_{D2B} + R_{22A}^2C_{TB}. \quad (23)$$

The corresponding formulas for WB are even simpler:

$$C_T = C_{TA} \quad (24)$$

$$T_P = T_{PA} \quad (25)$$

$$R_{22} = 0 \quad (26)$$

$$T_{D2} = 0 \quad (27)$$

$$T_{R2}R_{22} = 0. \quad (28)$$

A set of APL functions which implement this approach appear in Figures 8 and 9. The necessary data is passed around in the form of vectors. A two-port network is represented by the vector  $C_T$ ,  $T_P$ ,  $R_{22}$ ,  $T_{D2}$ ,  $T_{R2}R_{22}$ . The listing of WC, for example, shows the calculation of the required output, term by term, from the arguments. This function can be compared with (19) to (23).

Figure 9 shows five functions intended to calculate the bounds for any network. The two functions TMIN and TMAX calculate the lower and upper bounds for delay, and refer to a global variable named V which contains the threshold, a number (or array of numbers) between 0 and 1. The functions VMIN and VMAX calculate the lower and upper bounds for signal voltage and refer to a global variable T containing an array of delay times. The final function, OK, refers to both V and T and returns 1 if all is well, that is, if  $TMAX \leq T$ , or -1 if the network definitely will fail, that is if  $T < TMIN$ , or 0 if the bounds are not tight enough to tell for sure, that is if  $TMIN \leq T < TMAX$ . An example of the use of these functions to test the network in Figure 7 is shown in Figures 10 and 11.

Because these functions were written for exposition, no protection is included against meaningless values of V or T. In addition, these fail for networks without any resistances or capacitances, and for  $V = 0$  or  $T = 0$ .

#### V. Application to PLA Speed Estimates

These bounds are applied, as an example, to polysilicon lines driving the AND plane of a PLA, to determine whether or not the dominant delay occurs here. It is assumed that a strong superbuffer driver drives the line, and that every second minterm has a transistor present. The gates are assumed to be 4 microns square, separated by 24 microns of RC line. The poly resistance is assumed to be 30 ohms per square, the gate-oxide thickness 400 Angstroms, and the field-oxide thickness 3000 Angstroms.

These numbers lead to a capacitance of 0.01 pF and resistance 180 ohms between gates, and a resistance of 30 ohms and capacitance of 0.013 pF for each gate. The network is driven by a source resistance of 380 ohms and the effective capacitance of the output of the driver is estimated as 0.04 pF.

```

      ▽ Z←URC X
[1]   Z←X[2],(X[1]×X[2]÷2),X[1],(X[1]×X[2]÷2),X[1]×X[1]×X[2]÷3
      ▽

      ▽ Z←WB A
[1]   Z←A[1 2], 0 0 0
      ▽

      ▽ Z←A WC B
[1]   Z←(A[1]+B[1]),(A[2]+B[2]+A[3]×B[1]),(A[3]+B[3]),A[4]+B[4]+A[3]×B[1]
[2]   Z←Z,A[5]+B[5]+(2×A[3]×B[4])+A[3]×A[3]×B[1]
      ▽

```

Figure 8. APL functions for the element and wiring functions.

```

      ▽ Z←VMIN A
[1]   Z←(T≥A[2]-A[5]÷A[3])×1-(*(T-A[2]-A[5]÷A[3])÷A[2])×A[4]÷A[2]
[2]   Z←0[Z[1-A[4]÷T+A[5]÷A[3]]
      ▽

      ▽ Z←VMAX A
[1]   Z←(1-(A[4]÷A[2])×*-T÷A[5]÷A[3])[(T+A[2]-A[4])÷A[2]]
      ▽

      ▽ Z←TMIN A
[1]   Z←-⊗A[2]×(1-V)÷A[4]
[2]   Z←0[(Z×A[5]÷A[3])[A[4]-A[2]×1-V]
      ▽

      ▽ Z←TMAX A
[1]   Z←(A[4]÷1-V)-A[5]÷A[3]
[2]   Z←Z[(A[2]-A[5]÷A[3])-0[A[2]×⊗A[2]×(1-V)÷A[4]]
      ▽

      ▽ Z←OK A
[1]   Z←(T≥TMAX A)-T<TMIN A
      ▽

```

Figure 9. Response functions.

A AN EXAMPLE OF THE USE OF THE RC TREE DELAY CALCULATIONS.

BRANCH ← WB (URC 8 0) WC URC 0 7

NET ← (URC 15 0) WC (URC 0 2) WC BRANCH WC (URC 3 4) WC URC 0 9

A NOW THE NETWORK IS DEFINED.

V← 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9

A NOW THE VECTOR OF THRESHOLD VOLTAGES IS DEFINED.

A NEXT TO FIND THE MINIMUM AND MAXIMUM BOUNDS FOR DELAY:

V, (TMIN NET), [1.5] TMAX NET		
0.1	0	68.167
0.2	27.8	117.22
0.3	71.46	173.17
0.4	123.13	237.76
0.5	184.23	314.15
0.6	259.02	407.65
0.7	355.45	528.18
0.8	491.34	698.07
0.9	723.66	988.5

A NOW TO DEFINE A DELAY VECTOR AND GET THE VOLTAGE BOUNDS:

T← 20 40 60 80 100 200 300 400 500 1000 2000

T, (VMIN NET), [1.5] VMAX NET		
20	0	0.18138
40	0.03243	0.22912
60	0.0814	0.27565
80	0.12565	0.31761
100	0.16644	0.35714
200	0.34342	0.52297
300	0.48283	0.64603
400	0.59263	0.73734
500	0.67913	0.8051
1000	0.90271	0.95615
2000	0.99105	0.99778

Figure 10. Example of the use of the fast calculation scheme to find upper and lower bounds on delay and response voltage.

A function which returns a network with  $N$  minterms is shown in Figure 12. The results of calculating the delay as a function of the number of minterms are shown in Figure 13. The voltage threshold was taken to be  $0.7$  times  $V_{DD}$ . On this log-log plot the quadratic dependence of delay on number

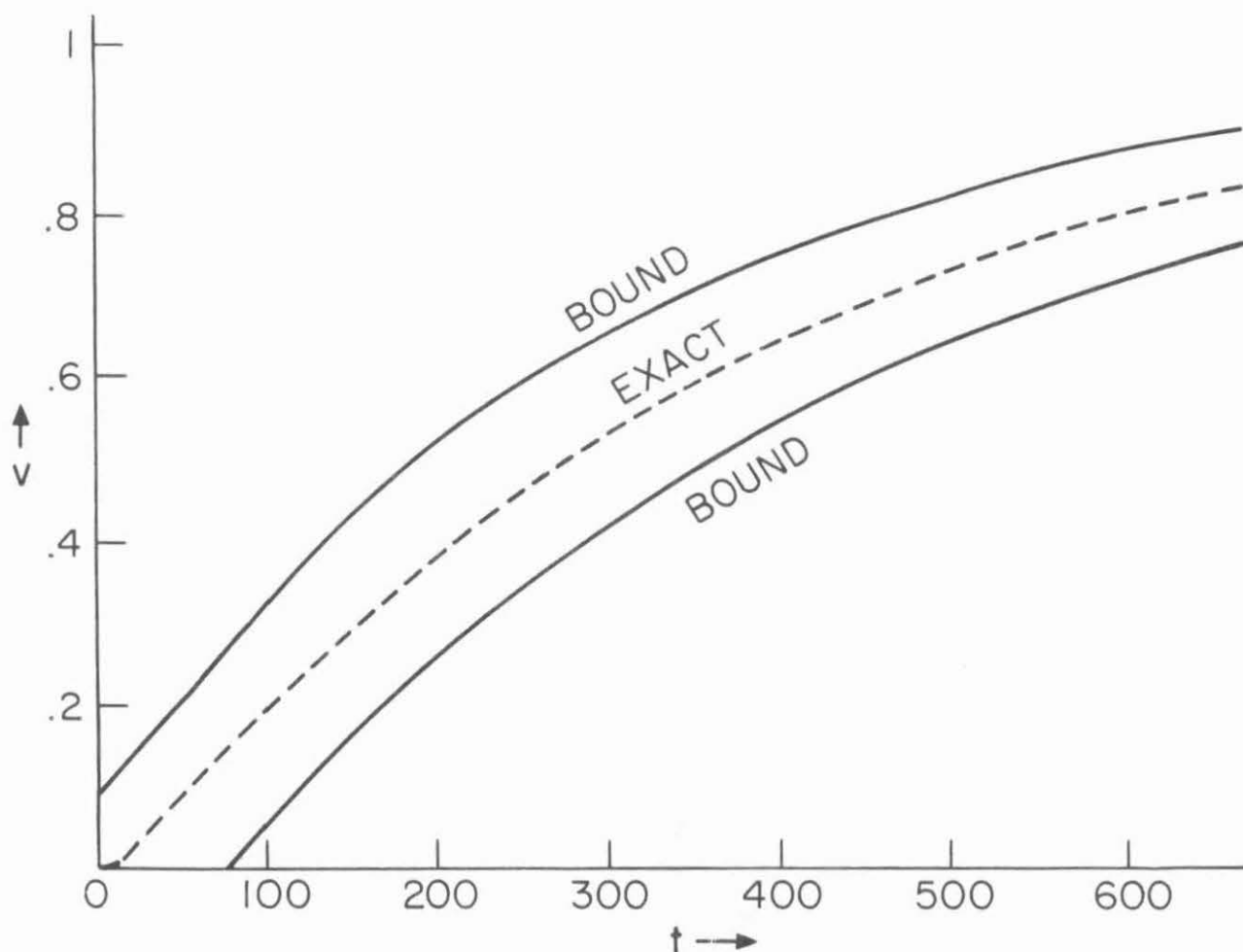


Figure 11. Upper and lower bounds as calculated in Figure 10. The exact solution, found from circuit simulation, is shown also.

```

▽ Z←PLALINE N;A
[1] A←(URC 180 0.0107)WC URC 30 0.0134
[2] A A IS A SINGLE SECTION ACCOUNTING FOR TWO MINTERMS
[3] Z←(URC 378 0)WC URC 0 0.04
[4] A Z IS THE PULLUP R AND C FOR SUPERBUFFER DRIVER
[5] LOOP:→(N≤0)/0
[6] Z←Z WC A
[7] N←N-2
[8] →LOOP
▽

```

Figure 12. APL function which returns a model of a PLA line with N minterms.

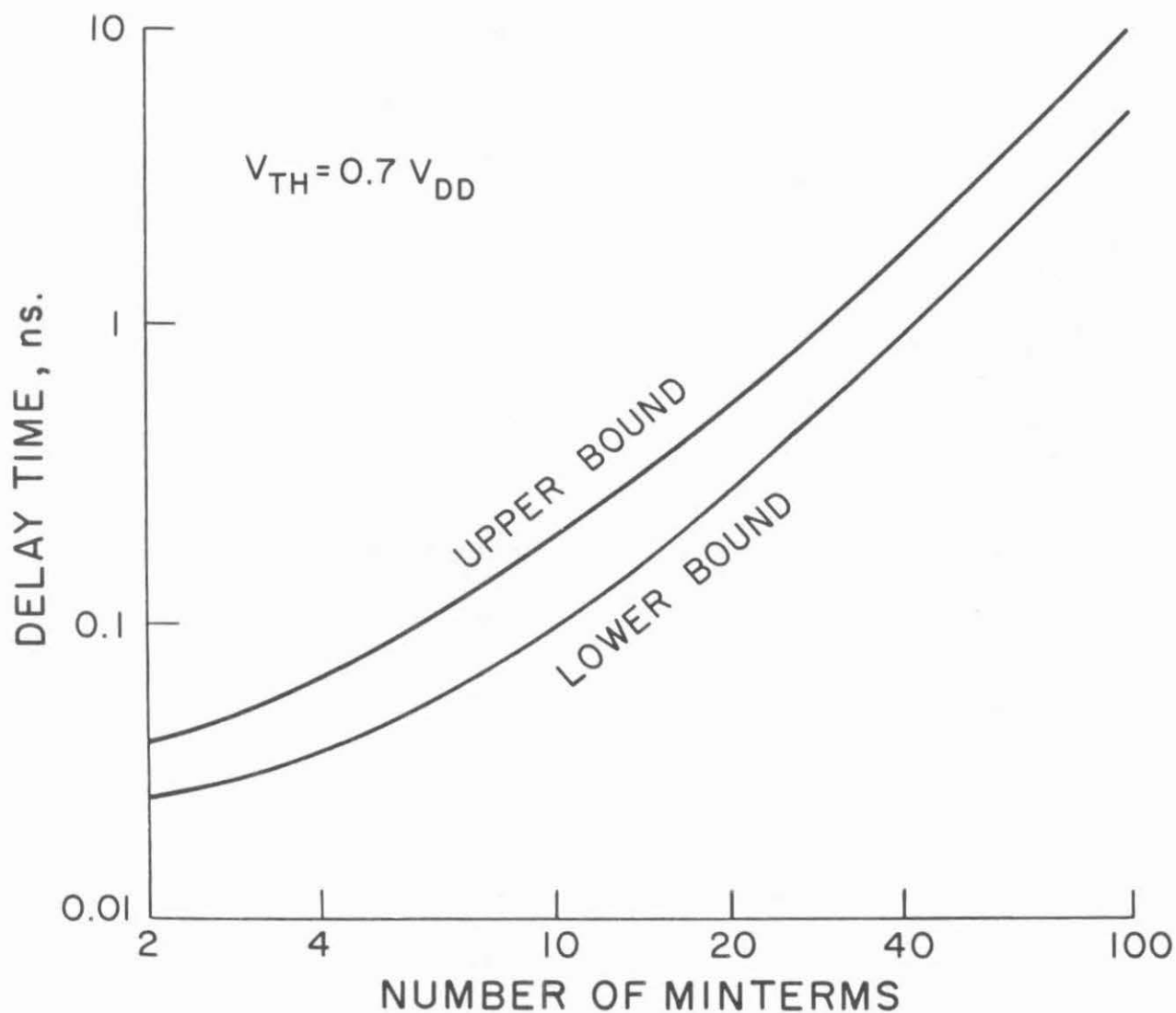


Figure 13. Upper and lower bounds on response time of the network of Figure 12, shown as a function of the number of minterms in the PLA.



of minterms (as a measure of the length of the line) is evident. Also evident is the fact that even with as many as a hundred minterms, the delay is guaranteed to be no worse than 10 nsec. This suggests that the dominant delay in a PLA occurs elsewhere.

## VI. Conclusions

A computationally efficient method for calculating the signal delay through MOS interconnect lines with fanout has been described. Tight upper and lower bounds for the step response of RC trees have been presented, together with linear-time algorithms for these bounds from an algebraic description of the tree. Substantial computational simplicity is achieved even in the presence of RC distributed lines by representing the RC tree by a small set of suitably defined characteristic times, which can be calculated by inspection and used to generate the bounds.

Although only the step response is considered here, the results can be extended to upper and lower bounds for arbitrary excitation by use of the superposition integral [1].

Extensions of the theory to RC trees with nonlinear elements (similar to the work of Glasser [3] for nonlinear MOS inverters) would be desirable for better modeling of MOS circuits. Investigations of RC trees with nonlinear capacitors and resistors are now under way, along with attempts to unify the modeling of gates and interconnects, and in particular to include pass transistors in the interconnects. Tighter bounds are also being looked for.

## VII. Acknowledgements

The authors are pleased to acknowledge discussions with Steven Greenberg, Llanda Richardson, and Lance Glasser, and help from Barbara Lory in manuscript preparation.

## References

- [1] J. Rubinstein and P. Penfield, Jr.; to be published.
- [2] W. C. Elmore, "The Transient Response of Damped Linear Networks with Particular Regard to Wide-Band Amplifiers," *Journal of Applied Physics*, vol. 19, no. 1, pp. 55-63; January, 1948.
- [3] L. A. Glasser, "The Analog Behavior of Digital Integrated Circuits," private communication; December, 1980.



# FUNCTIONAL VERIFICATION IN AN INTERACTIVE SYMBOLIC IC DESIGN ENVIRONMENT

Bryan Ackland  
Neil Weste

Bell Laboratories  
Holmdel, New Jersey 07733

## ABSTRACT

This paper describes verification techniques that have been implemented as part of an interactive symbolic IC design system. Circuit analysis programs perform node extraction and gate decomposition. They generate both transistor and gate level circuit descriptions which are used as input to a transistor level digital MOS timing simulator. The extraction programs make use of an intermediate circuit description language which captures both geometric placement and circuit connectivity. All programs are written in the *C* programming language and run under the *UNIX* operating system. An example is included to demonstrate the operation of these various techniques.

## 1. INTRODUCTION

Functional verification is an important and necessary step in the design of large scale integrated circuits. It is that part of the design cycle which eliminates most, preferably all, of the human errors introduced in the forward part of the design. It is generally a two stage process consisting firstly of automatic circuit extraction, in which electrical circuit descriptions are generated from the physical layout, and secondly of functional simulation of the derived circuit. Symbolic design techniques simplify the circuit extraction task as they introduce structural or circuit information into the layout file and remove unnecessary geometrical data. Interactive design techniques, however, place additional constraints on verification in that they demand fast response in order to avoid slowing the interactive design cycle.

This paper describes verification techniques that have been implemented on *MULGA* [1] - a *UNIX*<sup>†</sup> based interactive symbolic layout system. They consist of two programs which perform nodal circuit extraction and gate decomposition, and *EMU* - a transistor level MOS timing simulator. All programs are written in the *C* programming language and run under the *UNIX* operating system in a microcomputer based design station.

## 2. MULGA

Symbolic layout provides a means of abstracting the detailed and often laborious task of mask design. It offers the advantages of manual layout with regard to density, along with reduced design time and reduced likelihood of manual error. *MULGA* is a *UNIX* based interactive symbolic design system consisting of a suite of programs residing on a high performance color display station. Figure 1 shows the various software components of *MULGA* and

<sup>†</sup> *UNIX* is a trademark of Bell Laboratories

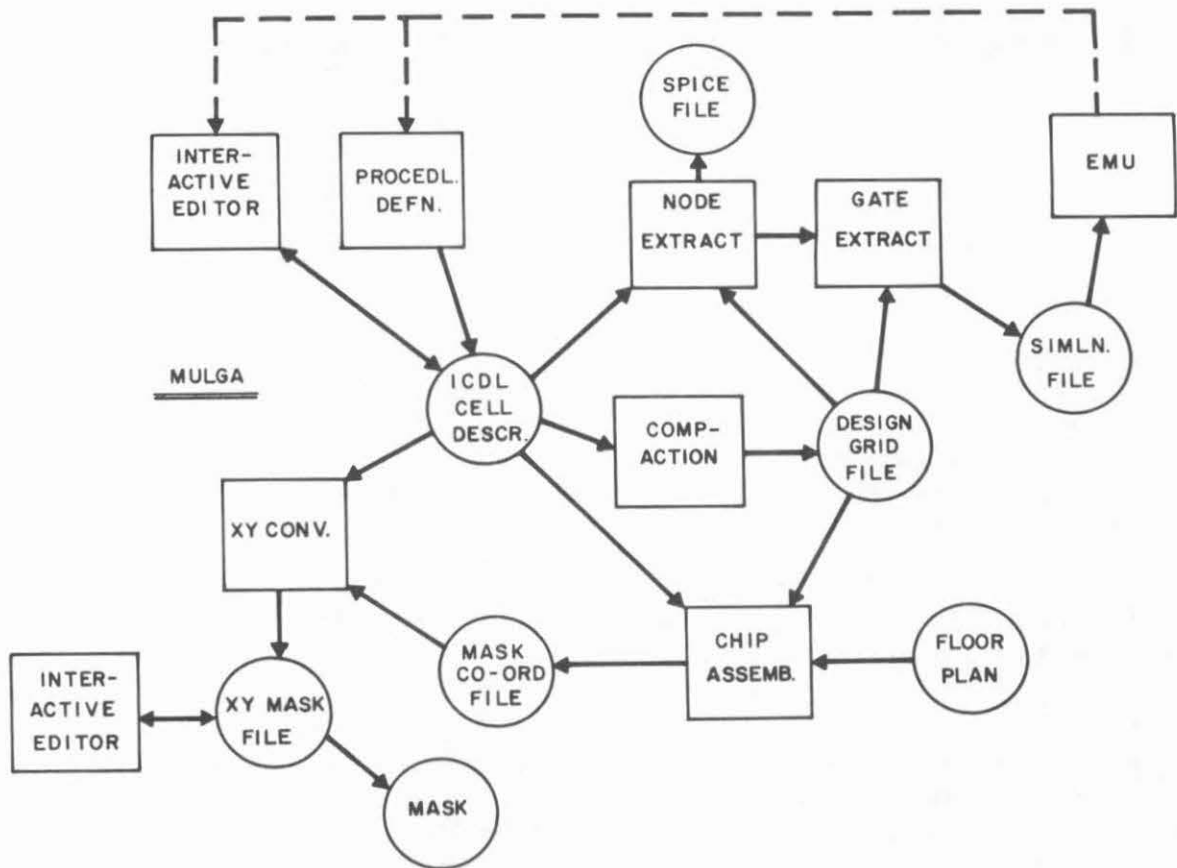


Figure 1. MULGA design system

the way in which they come together to effect a design.

The system is based around a symbolic Intermediate Circuit Description Language (*ICDL*) which uses a derivation of the co-ordinate notation introduced by Buchanan and Gray [2]. It combines circuit topology with geometric placement on a coarse virtual grid. In this way, the language captures designer intent with respect to the circuit, rather than a collection of abstract geometric forms.

The basic structure in *ICDL* is a cell which is a collection of elements placed on a virtual grid as shown in Figure 2. These elements may be devices, wires, contacts, pins, or other cell instances. Pins are named interconnection points that have no physical meaning in the final layout. They are a very important attribute of the language, however, and are used extensively in cell placement procedures and circuit verification. Figure 2 shows a *CMOS 2-input* nand gate represented graphically along side the textual *ICDL* description of the cell. Note that each line of text corresponds to an actual circuit component rather than a geometrical shape. Components are restricted to lie on grid intersection points as shown. Note, however, that this grid is only a relative placement network which defines the topology of the layout. Actual physical dimensions are determined later by a compaction process.

*ICDL* cell descriptions may be generated either via the interactive editor or else procedurally using the *C* programming language. Once the designer is satisfied with the symbolic

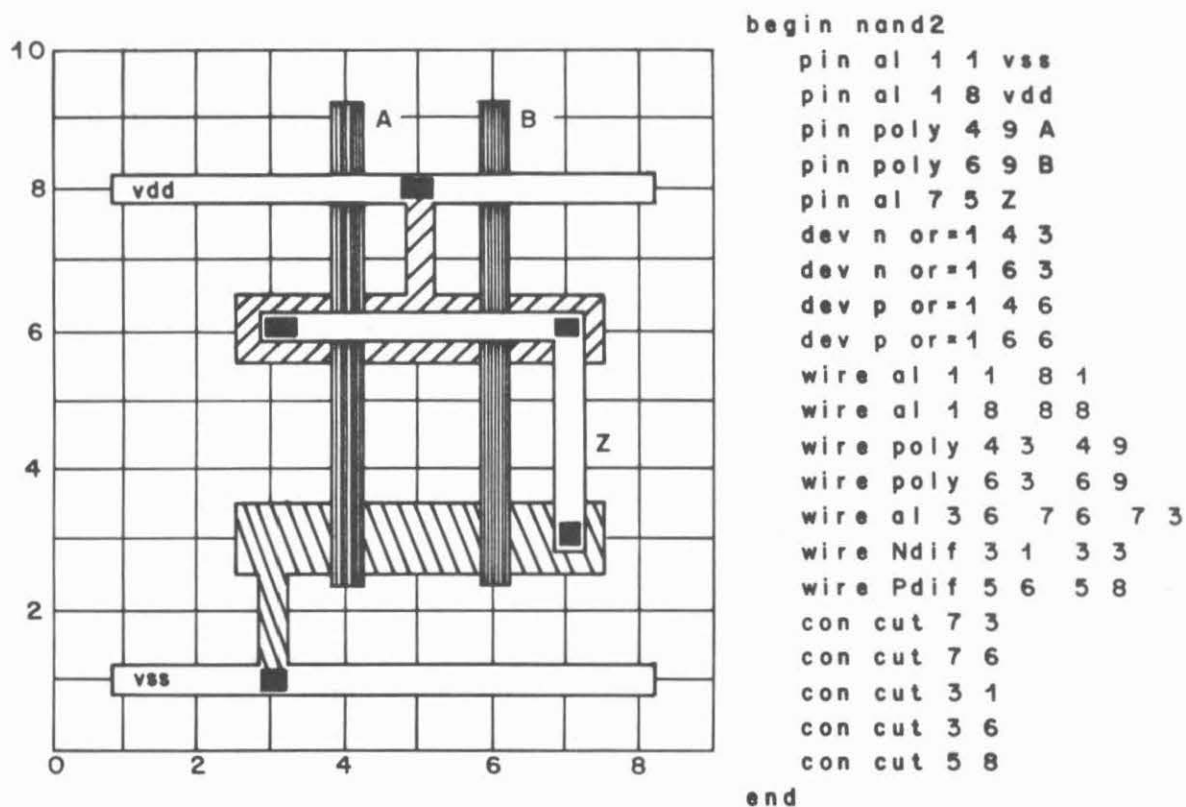


Figure 2. ICDL description of 2-input CMOS nand gate

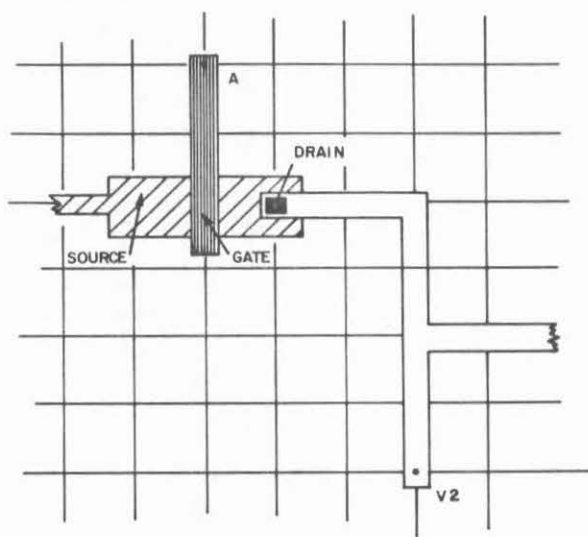
description, the file is compacted. The compaction program examines each symbolic grid line in the layout to determine how far it must be spaced from its neighbours in order to satisfy process design rules. Compaction information is stored in the design grid file. This file, along with the original ICDL description defines the minimum cell geometry assuming no other constraints in the design.

A chip assembler program takes the design grid file along with a specified chip floor plan and generates a mask coordinate file describing the actual physical location of the symbolic grid lines in the final layout. The chip assembler frequently needs to expand previously compacted cells in order to maintain global connectivity. The final step in the forward design path is the conversion and placement of cells into *XYMASK* data files. *XYMASK* is the geometric mask definition language used by the Bell System. A second interactive editor provides a means whereby the designer can view and evaluate his final design.

The verification phase consists of two circuit extraction programs and *EMU* - a transistor level MOS timing simulator. The first program performs node extraction and produces, in addition to the node list, a transistor level description of the circuit suitable as input to a circuit simulator such as *SPICE*. The second performs gate decomposition producing the higher level circuit description required by *EMU*. The following sections describe the operation of these three programs.

### 3. NODE EXTRACTION

The complexity of the circuit extraction process is heavily influenced by the nature of the layout definition language. One of the advantages of *ICDL* is that it carries implicit circuit connectivity information along with the physical topology. As shown in Figure 3, devices have designated connection points which are related by simple geometric rules to the center of the device. Wires serve to connect devices and external connections via interlayer contacts. Electrical connectivity is established when two elements exist on the same layer at the same virtual grid position. The pin construct aids the designer in naming specific nodes and connection points. This implicit connectivity is used, in conjunction with a simple algorithm, to arrive at a transistor node table description of the cell.



**Figure 3.** Implicit connectivity of *ICDL* components

The algorithm begins by first reading in a complete description of the *ICDL* cell. Following this, each pin, contact and transistor connection is assigned a different node number. Figure 4 shows a hypothetical net of components labelled in this manner. If there were no wires in the circuit, all such initial node numbers would be unique. Wires serve to connect components and reduce the overall number of unique nodes in the circuit. Accordingly, each wire is examined in turn to determine which nodes are redundant. A list is made of all nodes belonging to that wire. If the wire crosses another wire of the same type, connectivity is established by adding one node from the new wire to the old wire node list.

These wire node lists are used to eliminate redundant node numbers and generate a node net list description of the circuit. Pin names are used, wherever possible, to identify named nodes. Un-named nodes are given an internally generated name. Parasitic capacitance values for each node are calculated using the topology contained in the *ICDL* description along with absolute grid dimensions obtained from the mask coordinate file and specified process parameters. At this stage, sufficient information has been gathered to produce a transistor level circuit description. A simple filter converts this data into a *SPICE* simulation file. Figure 5 shows the simulation file generated from the 2-input nand gate described in Figure 2.

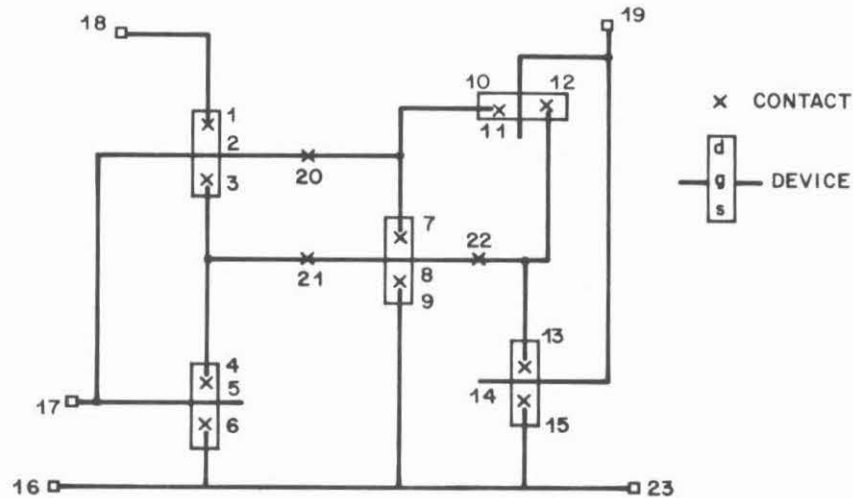


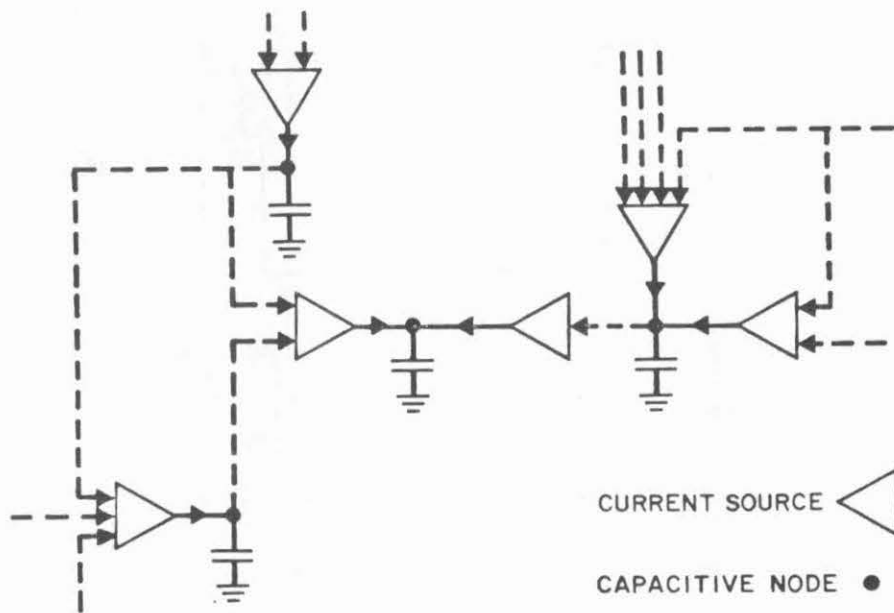
Figure 4. An example of initial node numbering

```
.SUBCKT nand2 ( vss vdd A B Z )
MN1 I0 A vss VSS N7A
MN2 Z B I0 VSS N7A
MP1 Z A vdd vdd P7A
MP2 Z B vdd vdd P7A
CMvss vss 0 CMTOSH 42
CNTvss vss 0 CN+H 12
CMvdd vdd 0 CMTOSH 42
CPTvdd vdd 0 CP+H 12
CPA A 0 CPTOSH 36
CPB B 0 CPTOSH 36
CMZ Z 0 CMTOSH 42
.FINIS
```

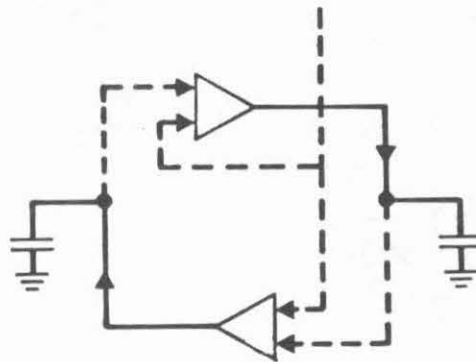
Figure 5. SPICE description of 2-input nand gate

#### 4. SIMULATION

Analog circuit simulators such as *SPICE* give very accurate reliable feedback as to the functional operation of a circuit. They tend, however, to be very expensive in terms of computer and engineer time. In an interactive design environment, ease of operation and fast turnaround are of paramount importance and some accuracy can often be sacrificed to achieve this. For these reasons, a *UNIX* based MOS timing simulator known as *EMU* was developed. An important feature of the simulator is the fact that it is a resident part of the design station software and is therefore capable of giving the designer fast feedback concerning the operation of his circuit.



(a) Generalized showing current sources and voltage nodes



(b) Bi-directional circuit element

**Figure 6.** EMU circuit model

Timing simulators fall somewhere in between circuit simulators and logic simulators. They model digital circuits as collections of idealized transistors which may be grouped in a defined manner to form simple logic functions. Unlike logic simulators, they generate an analog waveform and are able to deal with limited analog effects such as charge storage and bidirectional circuit elements. Performance, however, is typically one to two orders of magnitude faster than analog circuit simulators. *EMU* is a MOS timing simulator which, like *MOTIS* [3], uses certain properties of the MOS transistor to greatly simplify the circuit model. These properties are represented by the following approximations:



1. The input resistance of the gate terminal is infinite, i.e. the input impedance is purely capacitive.
2. The leakage current of a MOS device is zero.
3. The channel may be represented as a voltage controlled d.c. current source.
4. In a self-aligned digital process, Miller effects are negligible.
5. The impedance to ground at any node is dominated by diffusion, gate and wiring capacitances which are voltage independent.

These approximations lead to the model shown in Figure 6(a). The circuit consists of a number of capacitive nodes interconnected by various voltage controlled current sources. Any number of current sources may drive a single node. Bidirectional circuit elements are represented by two sources as shown in Figure 6(b). Current sources may be transistors, load devices or compound gate structures.

#### 4.1 Compound Gates

MOS gates typically consist of driver transistors connected in series/ parallel combinations as shown in Figure 7. Parallel branches pass current if any of the component elements conduct. This is equivalent to an OR function. Similarly, series branches conduct only if all component elements conduct - hence an AND function.

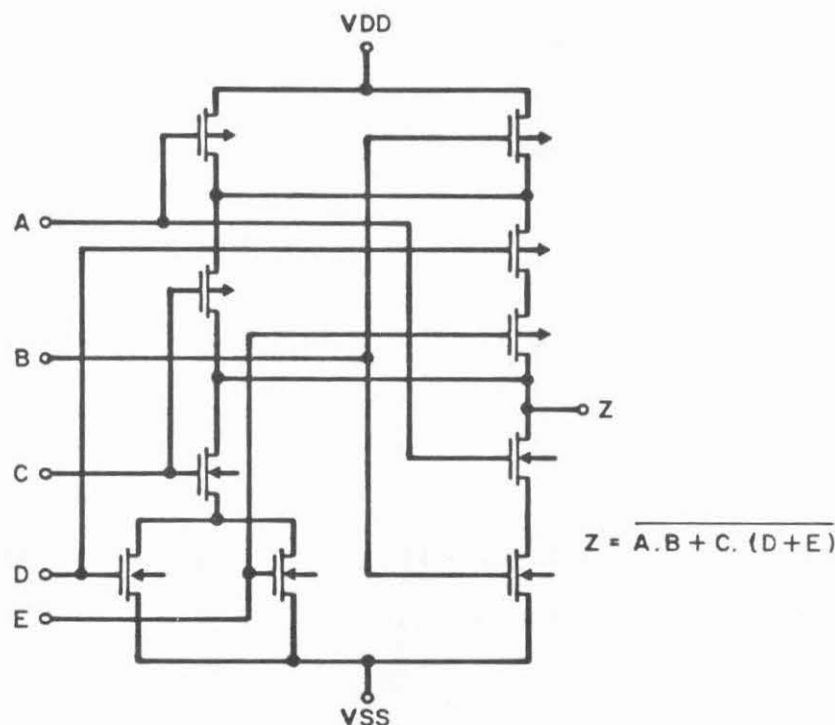
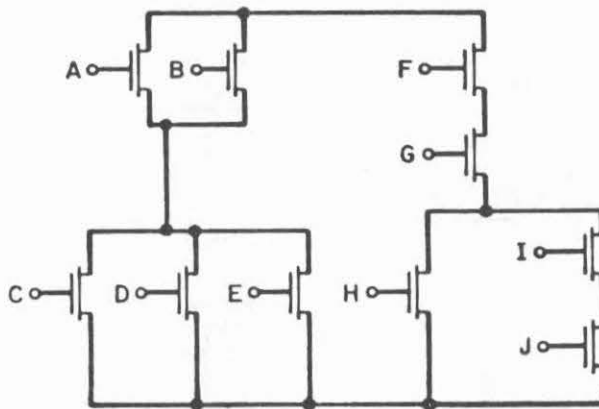


Figure 7. Typical CMOS gate construction

Although a logic gate can be simulated as a collection of individual transistors, much data space and simulation time can be saved if it can be modelled as a single current source. *MOTIS* uses the concept of a compound gate in which series/ parallel combinations of driver transistors are lumped together into a single data structure. It is based on the following approximations:

1. The total current sourcing a node is the sum of the currents sourced by all parallel branches connected to that node.
2. The current sourced by a branch containing several elements in series is the inverse sum of the currents that would be sourced by each element if it were the only element in the branch.
3. All transistors in the gate structure operate independently. Their only point of interaction is through the gate output node.

*EMU* uses this same compound gate construction. Driver transistors are specified in a "reverse polish" manner as shown in Figure 8. Transistor position in the gate is defined by the operators **push**, **parallel** and **series**. **Push** places a transistor on to an imaginary stack. **Parallel** means that the named device is in parallel with the top element of the stack. The resulting parallel combination replaces the element on the stack. **Series** means that the named device is in series with the top element of the stack. The resulting series combination replaces the element on the stack. **Parallel** or **series** without an operand, operates on the top two elements of the stack and pops the stack one position.



PUSH (A). PARL (B). PUSH (C). PARL (D). PARL (E). SERS.

PUSH (F). SERS (G). PUSH (I). SERS (J). PARL (H). SERS. PARL

**Figure 8.** Reverse Polish gate specification

This data structure is simply interpreted by a "reverse polish" current calculator. The calculator uses a real stack to store transistor conduction currents. The operator **push** causes a new current to be pushed on to the stack. The operator **parallel** causes two currents to be added. Similarly, the operator **series** causes two currents  $I_a$  and  $I_b$  to be combined according to:

$$\frac{1}{I} = \frac{1}{I_a} + \frac{1}{I_b}$$

## 4.2 Gate Advancement

Gate advancement is the technique by which node voltages are updated for each new time step of the simulator. Referring to Figure 6, each node in the circuit model is dominated by a capacitance to ground  $C$ . This means that for a sufficiently small time step, node voltages within the circuit are essentially constant. Source currents, which are in turn functions of circuit voltages are therefore also constant. Suppose a node is driven by  $n$  current sources  $I_1, \dots, I_n$ . The total current into the node is then  $I = \sum_{k=1}^n I_k$ . For a sufficiently small time step  $(t - t_0)$ , the new node voltage  $V(t)$  is given by:

$$v(t) = V(t_0) + \frac{I(t - t_0)}{C}$$

The accuracy of this simple forward integration scheme depends critically on the choice of time step. If the time step is too large, circuit voltages and currents may change significantly during one iteration and errors will be introduced. In addition, voltages tend to overshoot leading to numerical instability - especially with bidirectional circuit elements such as transmission gates. On the other hand, if the time step is too small, much simulation time is wasted iterating over unnecessarily small time intervals.

Rather than leaving this delicate choice of time step to the operator, *EMU* automatically adjusts the time step to maintain simulation accuracy. It does this by monitoring the maximum voltage step occurring from one time instant to another and adjusting the timestep accordingly. Simulation thus proceeds rapidly during periods of low circuit activity and then slows down to critically examine those periods when changes are taking place.

## 4.3 Device Model

The basic Sah model is used to calculate MOS transistor channel current. The equations, as applied to an N channel device are:

$$\text{Cutoff:} \quad |V_{GS}| < V_t$$

$$I_{DS} = 0$$

$$\text{Non-saturation:} \quad |V_{GS} - V_t| > |V_{DS}|$$

$$I_{DS} = \beta \left( \frac{w}{l} \right) \left[ (V_{GS} - V_t) V_{DS} - \frac{(V_{DS})^2}{2} \right]$$

$$\text{Saturation:} \quad |V_{GS} - V_t| \leq |V_{DS}|$$

$$I_{DS} = \beta \left( \frac{w}{l} \right) \frac{(V_{GS} - V_t)^2}{2}$$

where  $w$  and  $l$  are the channel width and length respectively.

Back-gate bias effects are taken into account using table lookup techniques to calculate perturbations in threshold voltage.

## 4.4 OPERATION

The operation of *EMU* is characterized by four software states. Initially, *EMU* enters the **command** state. This is the common state from which all others can be entered. It is used to set simulation parameters, define clocks, display portions of the data base, initialize inputs and format the output of results. The **circuit** state is used to create a circuit description in the data

base. It is used to define inputs and nodes, assign gates and set circuit capacitances. The **process** state is used to set process parameters such as transistor threshold voltage and transistor gain. The **execute** state represents the actual simulation. Following execution, the simulator returns to the **command** state. **Command**, **input** and **process** states each have their own input language which may be entered interactively or via an predefined input file.

#### 4.5 Performance

*EMU* is written in *C* and will run on any *UNIX* based machine. In particular, it runs on the *LSI-11/23* - the host processor in the *MULGA* design station. It has also been implemented, however, on a *VAX-11/780* and a Motorola 68000. The 11/23 implementation occupies approximately 25K bytes of code space leaving 30K available for circuit definitions. This is sufficient to hold a circuit of about 2000 transistors. Table I shows some simulation run times for a sample circuit - a 32×1 bit CMOS static RAM. This circuit contains 260 gates which in turn contain some 770 transistors. Note that even on the 11/23 microcomputer, this type of circuit can be simulated in a time which compares favorably with the time needed to perform an off-line simulation on a larger machine.

SIMULATION RUN TIMES (secs)	
LSI-11/23	1094
VAX-11/780	129
68000 (4 mHz)	1010
68000 (8 mHz)	585

TABLE I SAMPLE RUN TIMES

#### 5. GATE DECOMPOSITION

The nodal analysis program described in Section 3 produces a transistor net list which can be used to generate a transistor level circuit description for *EMU*. The speed of the simulator, however, is directly related to the number of active current sources in the circuit. Accordingly, a gate extraction program has been written to process this transistor net list and convert it, where possible, into the compound gate structures recognized by *EMU*.

As a first step, nodes are characterized as either inputs, outputs or internal nodes. Outputs are defined to be those nodes which connect (in the case of a CMOS design) to the drains of both an N and a P transistor. Inputs are those nodes which connect only to transistor gate terminals within the cell. All remaining nodes are assumed to be internal. The algorithm then examines each output node in turn, and searches for all N devices connected either directly or indirectly (through an N channel) to that node. Any branches that pass through other output nodes are assumed to contain transmission gates and are ignored. All such N branches must eventually terminate at the negative supply rail if they are indeed part of a compound gate structure. Branches that do not satisfy this condition are discarded. This is equivalent to traversing the graph of all potential gate transistors connected to the output node. Figure 9(a) shows a number of devices connected to an output node Z. The N transistor graph that is derived from this circuit is shown in Figure 9(b). Each device is represented by a simple PUSH operator, consistent with the notation described in Section 4.1.

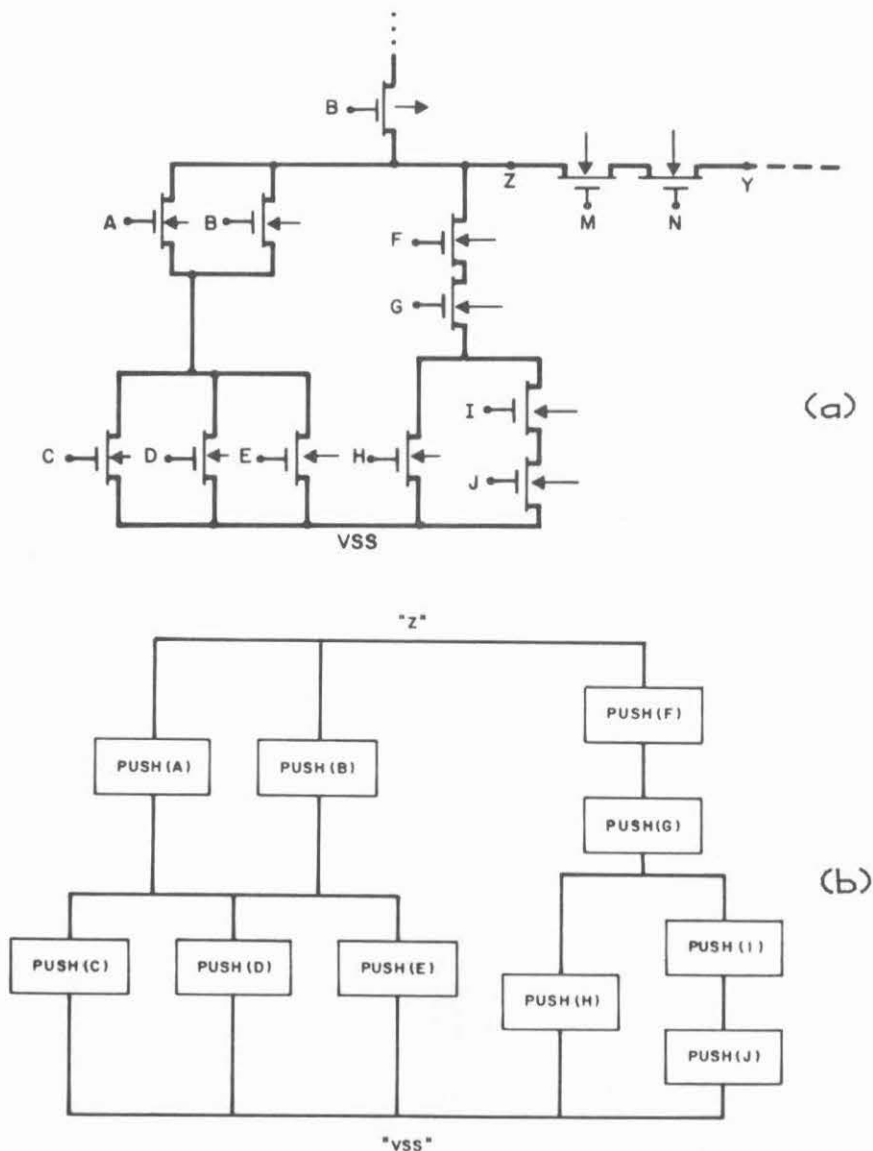


Figure 9. Example of gate device graph extraction from a circuit

The gate transistor graph is then reduced by successive parallel and series merging until a single branch remains. At each merge, the appropriate operator (SERIES or PARALLEL) is added to the branch description. The final result is the desired reverse polish specification of the transistor graph. Figure 10 shows the four steps required to reduce the graph of Figure 9.

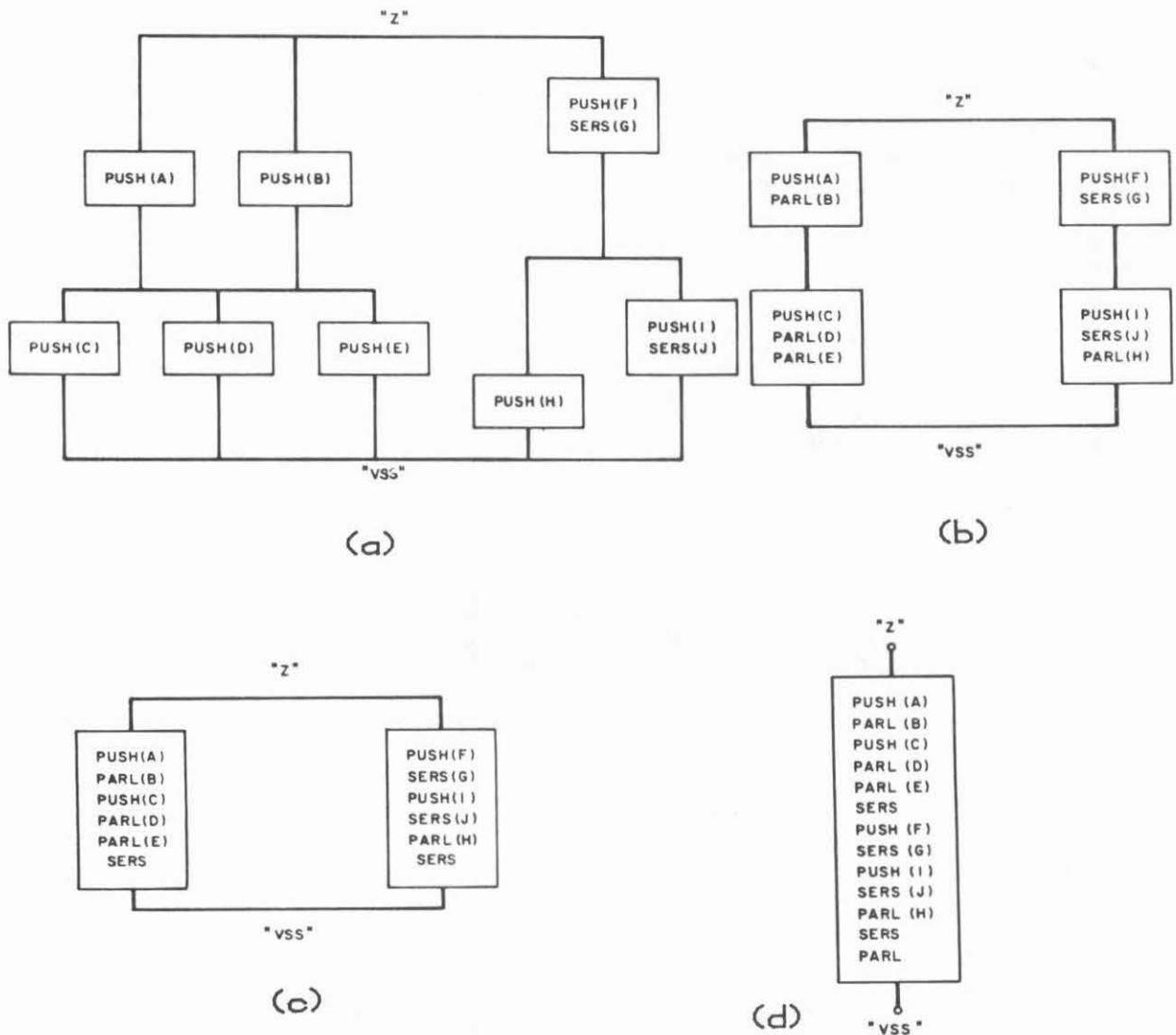


Figure 10. Series/parallel reduction of device graph

This process is then repeated for the P transistor chain. The two expressions are then combined to produce a complete gate description. Transistors not absorbed by this extraction process are retained as single transistor transmission gates. Figure 11 shows the gate level description that was extracted from the 2-input nand gate example of Figure 2. This description was given to *EMU* and Figure 12 shows the actual simulation result.

```
nand2 (A, B, Z)
EXTERN A, B;
EXTERN Z;
{
    GATE (Z, 5);
        PUSH (A, 1); SER (B, 1); PCH ();
        PUSH (B, 1); PAR (A, 1);
    INCAP (A, 30);
    INCAP (B, 30);
    OUTCAP (Z, 70);
    ROUCAP (Z, 9);
    ROUCAP (A, 11);
    ROUCAP (B, 11);
}
```

Figure 11. Gate level description of 2-input nand gate

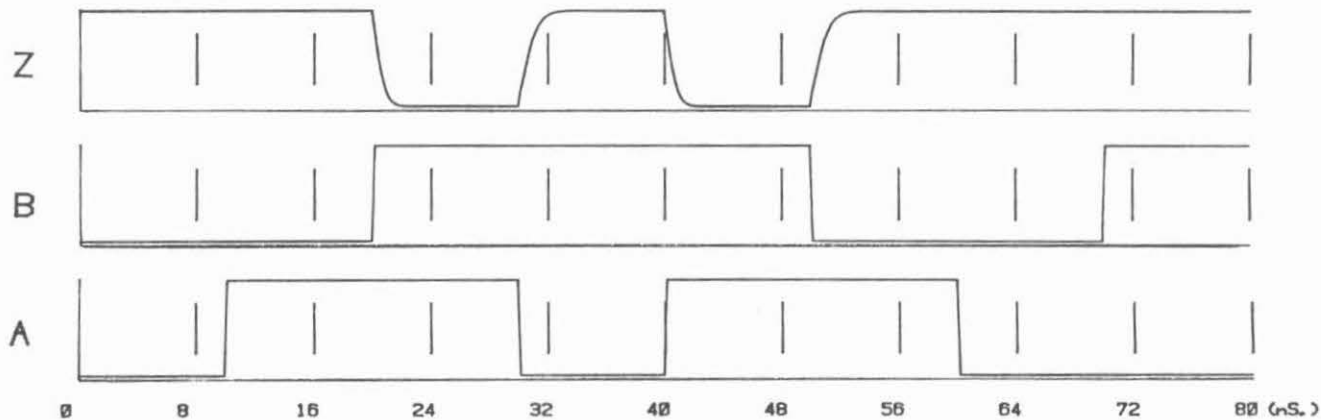


Figure 12. Simulation plot of 2-input nand gate

## 6. CONCLUSIONS

In an interactive design environment, verification tools must be fast and readily accessible in order to encourage the designer to perform this vital task. This paper has described three tools which meet these criteria by actually being part of the resident design station software. In addition, they have the advantage of being written in *C* and *UNIX*, making them readily transportable to other design systems.

## REFERENCES

- [1] WESTE, N., "MULGA - An Interactive Symbolic System for the Design of Integrated Circuits", *Bell Sys. Tech. Journal*, to be published.
- [2] BUCHANAN, I., "Modelling and Verification in Structured Integrated Circuit Design", *PhD Thesis, University of Edinburgh, Scotland*, 1980.
- [3] CHAWLA, B.R., GUMMEL, H.K., and KOZAK, P., "MOTIS - An MOS Timing Simulator", *IEEE Trans. on Circuits and Systems*, Vol. 22, No. 12, Dec. 1975, pp. 901-910.



A METHODOLOGY FOR IMPROVED VERIFICATION  
OF VLSI DESIGNS WITHOUT LOSS OF AREA

Louis K. Scheffer  
Departments of Electrical Engineering  
and Computer Science  
Stanford University  
Stanford, California 94305

ABSTRACT

This paper describes an IC layout methodology based on arbitrary outline cells, prevention of overlap, and mixed programs and graphics. Advantages are: no loss in area over hand packing; incremental checking of design rules, component interconnection, and timing; reduction of visible complexity; and easy implementation. Disadvantages are: possible proliferation of cell types and poor handling of cells with contacts not on the boundary. An implementation that uses and enforces this methodology is discussed.

INTRODUCTION

IC design methodologies in use today have a number of serious defects with respect to design verification. They defer design rule checks (DRC) and electrical continuity tests until the end of the design cycle. This is necessary since the possibility of another item overlaying an item that has already been checked cannot be ruled out. However, at the end of the design cycle, errors that could have been fixed quite simply earlier may require extensive revision. Multiple errors may require multiple analyses of the entire chip to find all the errors. The classic example of this is the power ground short. If extracting a component list from the artwork reveals a power to ground short, then the list will be useless for other purposes such as simulation or error checking. It is necessary to fix the short, and rerun the component extraction program on the entire chip. Furthermore, DRC and component extraction require execution times ranging from  $N \log N$  to  $N^{3/2}$ , making these tests time consuming and expensive as chips increase in size.

In this paper, a methodology is introduced that depends on not allowing cells to overlap other cells or geometrical primitives, but allows cells to have arbitrary shapes. The advantages and disadvantages of this methodology

are discussed, and comparisons made between this methodology and others that might be adopted to solve the same problems. A discussion of how the problems of DRC and component extraction are treated in this system follows, along with a proof that shows that this methodology does not cost any area. Next there is a discussion of the relationship between this methodology and schematic drawing systems, and a section on an implementation of an IC layout editor that uses and enforces this methodology. Finally, there is a section on possible future developments using this methodology.

#### BASIC IDEA OF THE METHODOLOGY

The theme of this methodology is simply to avoid overlap. Towards this end, cells may not overlap each other, and cells may not overlap geometrical primitives, such as rectangles and polygons. The only objects that may overlap are geometrical primitives within the same cell. Cells, however, may be of arbitrary polygonal shape. To replace the capability lost when overlapping cells are outlawed, a mechanism is supplied to mix layouts and programs to define devices such as ROMs and PLAs. In addition there is a requirement that no active component area touch the boundary of a cell.

An analogy exists between this methodology and that of structured programming. In both cases, the intent is reduce the interaction of disparate parts of the system, so that each part may be designed, tested and understood by itself, independent of the remainder of the design. Both methodologies attempt to hide the details of the implementation; in programming the essential information is contained in the calling sequence, whereas in IC layout it is contained in the boundary of the cell. Not allowing overlap is similar to not allowing GOTOs into or out of procedures. Not allowing components to touch the boundary of a cell is similar to not allowing statements to be split over procedure boundaries.

This analogy explains why verification is so much easier if overlap is forbidden. Since there is no equivalent of global variables in IC design, each piece may be considered on its own, just as a procedure that has no global variables may be checked on its own. Furthermore, a designer (or a program) may use a cell without knowing anything about it except the behavior as expressed from the terminals.

One final analogy is that when GOTOs are eliminated, a richer variety of control structures is required if efficiency is not to be sacrificed. Similarly, in IC design, if overlap is not allowed then cells must be allowed to assume arbitrary shapes.

#### ADVANTAGES AND DISADVANTAGES OF THIS METHODOLOGY

There are several advantages to this choice of methodology. First, it does not cost any area over a hand layout (proof follows). Second, it allows a completely hierarchical design, where only two adjacent levels of the hierarchy need to be examined at any time. This reduces the complexity as seen by the user and any analysis programs. Third, many checks on the validity of the data may be made incrementally, as each cell is defined and used. These include DRC, component extraction, and timing verification (if

the technique of assertions is used). Many errors can therefore be caught at an early stage, where correction is still easy, instead of at the end of the design process where correction is quite difficult and costly. Fourth, most processes that must be applied to the entire chip, such as the generation of new layers through logical operations on the existing layers, may be done by processing the chip on a cell by cell basis. This results in large savings when cells are used more than once.

Another advantage of this methodology is that it makes minimal demands on computer resources. This translates into low cost per station, either with a large computer timeshared between several stations or with a small computer per user. Since all processing is done one cell at a time, large amounts of memory are not required in the processor that does the DRC and component extraction. By using assertions, the work necessary to verify performance can also be vastly reduced.

Naturally, there are disadvantages as well. One serious objection is that designers are not used to designing with a total lack of overlaps. If the advantages described above are not attractive enough, the designers will be reticent to use a system that takes away some of their former freedom. Another objection is that this methodology is optimized for cells with their connections on the edges. If cells have terminals in the middle then this methodology will require a large number of nearly identical cells, with the only differences being the wiring required to bring the internal terminals to the edge. A third problem may appear if an operation such as oversizing or undersizing a mask is attempted. In this case the modifications within a cell may depend on the surroundings. Thus these modifications cannot be done on a strictly cell by cell basis, and it may be necessary to accept a proliferation of nearly identical cells in order to perform these operations and still keep the results in the strictly hierarchical format. These problems should not be serious for highly structured chips, but may present real barriers to using this methodology in other cases. For example, it is very ill-suited to masterslice type construction.

There are other problems that this methodology does not help, but does not hurt either. These are primarily calculations of a global nature that cannot be completed until the entire circuit is known. The general problem of calculation of the resistance of interconnections and the resulting time delays is one of these problems. Since the problem is inherently global, unlike capacitance (which is the sum of the local capacitances), the entire net must be known before the resistance or time delay may be calculated. Therefore this cannot be done at cell definition time, but must wait until the cell containing the entire net is constructed.

Penfield and Rubinstein [Pe81] have recently demonstrated a method which bounds the time delay for tree structured networks. This case can be solved on a local basis, and hence fits very well with the proposed methodology. The only test that would still have to be made globally is to insure that the interconnections are indeed tree structured, since this cannot be determined from local data.

## COMPARISON WITH OTHER METHODOLOGIES

There are many other methodologies that may be used to create a chip. How does this methodology compare? Three strategies that have been considered are: allow any overlaps and do all checking at the end of the design process, allow overlaps but account for them correctly, and do not allow overlaps and use rectangular bounding boxes.

Currently, the strategy used by most artwork systems is to allow the designer to create any overlap desired. Design rule checks, and tests for logical correctness, are performed once the design has been completed and the entire design expanded out. This approach gives the designer the maximum amount of freedom, and makes editing very simple, since anything is allowed. However, it requires that the checking programs must run on huge amounts of data. In particular, repeated cells are checked as many times as they occur. As chips get bigger, the volume of data will only get larger. Furthermore, design errors are found at the end of the design cycle, when they are the most difficult to fix. Errors in repeated cells show up many times, making it hard to find errors that may only have occurred once. One error, such as shorting power and ground, may make it impossible to find others without repeating the entire cycle.

Another methodology, tried by Whitney [Wh80] for DRC, is to allow the designer to overlap items wherever desired. The resulting design is checked one cell at a time, but the design rule checking program detects and accounts for these overlaps in the checking process. This approach also puts no restrictions on the designer, and allows the traditional methods of programming PLAs and ROMs. Redundant errors are also reduced, since each cell is checked only once except where it overlaps others. The drawbacks are that checking must still be done on the whole design at the end of the cycle, and that accounting for the overlaps may take more computational effort than simply checking the entire design as one piece. Furthermore, this approach is not well suited to incremental checking, since there is no guarantee that any portion of the design is complete until the entire design is finished. This approach probably represents the best that can be done without imposing any restrictions whatsoever on the designer.

McGrath and Whitney [McG80] propose a methodology that allows cells to overlap, but requires that each cell be correct when checked by itself, and all active devices to be represented by specially marked cells. The requirement that each cell be correct by itself leads to many false errors in the cases where a half width feature on a boundary is mated with a similar feature on the other side of the boundary. The suggested solution is to include full width features, and avoid the loss of area by overlapping the two full width edges. This means that overlap must be permitted, which leads to the problems in DRC discussed under the last methodology. Component extraction must also take the overlap into account, at least for capacitance calculations. Isolating active devices in cells of their own is almost a necessity in bipolar technology, where a given geometry could be a resistor, a transistor, or a zener depending on how it is biased, but is a quite significant (and unnecessary) restriction on the designer in normal MOS design. The transistors may be recognized quite easily, and unintentional

transistors are caught when the layout is compared to the logic schematic.

A fourth methodology is followed by many automated layout programs such as CICLOPS [Pr79]. This is to allow no overlaps, and to require rectangular cell boundaries. This allows design and analysis to proceed on a cell by cell basis, but often wastes area since only rarely is it possible to combine several different sized rectangles into a resultant rectangle without any wasted area. Because of this obvious waste, human designers are reluctant to use this methodology, especially for high volume products.

#### NO LOSS OF AREA

It is straightforward to show that the proposed methodology results in no loss of area in any technology with only one active layer. Given any design, first expand out any hierarchy that may be present. At this point, the data is merely a large number of polygons with no included cells, and hence it meets all the requirements of the proposed methodology. Now iteratively perform the following procedure: cover the chip with two non-overlapping polygons in such a way that half of the devices are inside one polygon, and the other half of the devices are in the other. The polygon boundaries may not pass through any devices. Since the devices are topologically separate in a technology with one active layer, this division into two polygons can always be performed (although it may require using polygons with internal holes in pathological cases). Call each polygon that has been created this way a cell; then the chip is now represented as two cells, each of which meets the methodology requirements. These cells may be in turn subdivided further, until the entire chip is represented as a hierarchy, with each cell containing at most two other cells, or two devices.

It should be emphasized that although it is possible to automatically split a chip up this way for DRC purposes, it is far better to have people design hierarchically in the first place. If this is done, then no arbitrary splitting of any sort is necessary. Splitting a complete chip voids most of the benefits of the proposed methodology, and is only intended as a thought experiment to show that any design may be represented in this methodology without loss of area.

#### PERFORMING OPERATIONS IN THE PROPOSED METHODOLOGY

Design rule checking is done as follows in this methodology. Let MAXRULE be the maximum distance specified in any of the design rules. Define the OUTLINE of a cell as everything within MAXRULE of the boundary. Then, for each cell, substitute in the outlines of all the lower level cells, and perform all the DRC tests. Ignore all errors within the outline that could possibly be affected by whatever surrounds the cell, for it is not known whether these are errors until the cell is used. An error of this type is shown in figure 1, for the case of width violations. If the square C is a minimum size feature, then the rectangles A and B represent potential width violations. Rectangle A cannot be fixed by the addition of geometry outside the cell boundary, and hence represents a real error that must be reported. Rectangle B, however, could be fixed by the addition of another rectangle when the cell is used, and therefore it is not reported. If the additional



geometry is not added to a cell in which this cell is used, then the error will be reported at that time.

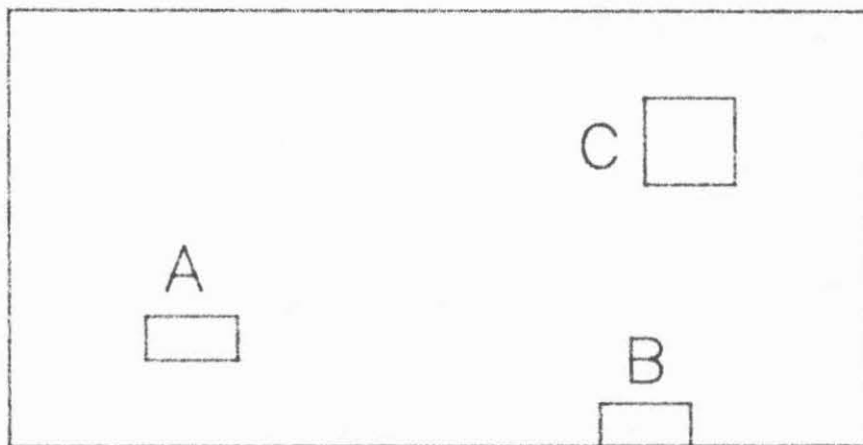


Figure 1: Real and potential errors

Similarly, the design rule checking code must ignore all errors that could be fixed by something inside the inner boundary of an included cell. If the error is real, it would have been caught when the cell was checked. This step is quite important, since the process of saving everything within MAXRULE of the boundary may split internal items into pieces that do not meet the design rules. Figure 2 shows the portion of a cell that is retained and substituted in wherever the cell is used. Since only the geometry within MAXRULE of the edges is retained, the polygon at A has been split, creating a possible error. The design rule checker, however, should not report this error, since it is against an inner boundary, and if it was a real error then it would have been caught when the cell was checked.

Any remaining errors are valid, since nothing can be added within MAXRULE of them, and should be displayed. The final step of the design rule checking process saves the outline, which will be substituted for the cell whenever the cell is to be involved in DRC checks.

This approach handles split contacts and half-width lines on the edges of cells without special cases. Furthermore, all of the rules, including the complicated anti-reflection rules, may be checked in this manner. The designer therefore gets immediate feedback as each cell is created or modified. This is important since errors are much harder to correct later on. The ability to check all rules is in contrast to some interactive design rule checkers that check for validity while the editing is in progress. These give even more immediate feedback, but cannot check some of the more difficult rules. The proposed methodology will never create false errors, but errors in the outline of the cell may not be caught until the cell is used, and may appear as many times as the cell is used.

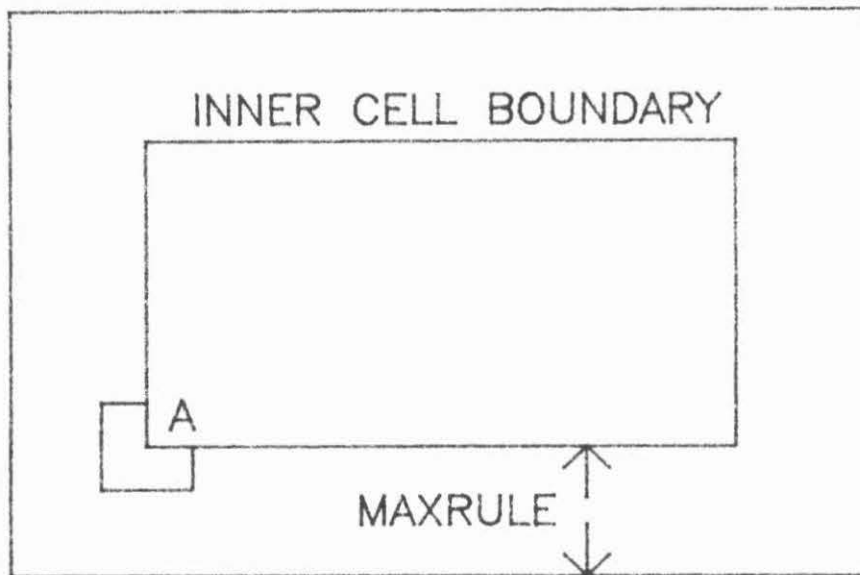


Figure 2- Possible internal error

Component extraction is made simple by the fact that no active areas may touch the boundary. This means each device is entirely contained within a cell, and only interconnections touch the edges. Implied devices, such as transistors formed where polysilicon overlaps diffusion, are not a problem since we know nothing will overlay the cell under consideration. Thus each cell can be extracted separately and reduced to a boundary that shows the connection points. Capacitance (as a function of area) may be easily calculated since the area of a node is the sum of the area in each of the cells that use that node (since there are no overlaps). Sidewall capacitance may also be computed, with the proviso that sidewall capacitance on the cell boundary cannot be computed until the cell is used. Similarly, node to node capacitance may be computed, at least for nodes lying within MAXRULE of each other. (MAXRULE could be increased if this proves to be a restriction.)

Enforcement of this methodology is quite easy. Check to see that no cells overlap, check to see that no primitives overlap any cells, and check to see that no components extend past the boundary, if the boundary is specified by the user.

Finally, some mechanism is needed to allow the operations that formerly were done by overlapping cells, such as filling out the bits in a ROM or PLA. One approach to this problem is to allow programs access to the graphics primitives, so a program can write a bit pattern to be included in a ROM or PLA. The program does not get any special privileges such as the ability to overlap cells. The output is subjected to exactly the same tests as if it had been entered by a human. This is useful for catching errors in the programs.

## RELATION TO SCHEMATIC DRAWING SYSTEMS

In many ways, the proposed methodology resembles the methodology used in schematic drawing systems. This is not a coincidence, since every attempt was made to incorporate the features that have made hierarchical schematic drawing systems useful. In a schematic, the user never views more than two levels of the hierarchy at a time. Any arbitrary structure can be represented, although some fit more naturally into the hierarchical structure than others. Individual pieces may be checked with the assurance that the rest of the design will not affect the piece that is being checked, except through the terminals of the device. These properties all carry over to layout, if it is done with this methodology.

In particular, two forms of checking originally designed for schematics are now possible for layouts. One is the timing check by means of assertions, as used by McWilliams [McW80]. In this technique the designer supplies statements about the signals which are believed to be true (the assertions). For example, consider the case of a simple cell with one input and one output. One assertion might state that the input must be true 50 NS after the clock, and another assertion might state that the output will be stable and correct by 90 NS after the clock. With the aid of these assertions, a cell may be checked without looking at the contents of any other cells.

In order to check a cell by this technique, the timing verifier assumes that the input assertions are correct, and attempts to prove the output assertions from the inputs and the properties of the devices contained in the cell. If any other cells are included in the cell under consideration, the verifier checks to be sure that their input assertions are met, and assumes that their output assertions true. By this means, an entire design may be verified one cell at a time, and timing verification may be done on the pieces long before the design is complete.

A second check that is easy to make using the proposed methodology is a comparison between the logic schematic of a cell and its layout. In the simplest form, the checking program could require that the hierarchy be identical in the schematic and layout representations. Then checking is quite simple. In more complicated forms, a theorem prover may be called into play when the correspondence is not exact, to see if the two different representations perform the same function. In either case, the check may be performed on a cell by cell basis.

There are still fundamental differences between a schematics and layouts, however. In a schematic, the inside and outside views are distinct. An op-amp, for example, is composed of a large number of transistors, but the configuration of these transistors bears no relation to the triangular symbol that represents the op-amp. Furthermore, the size and shape of the symbol do not depend on the actual complexity of the opamp, so the designer can design and wire an active filter, for example, without knowing the gain bandwidth products of the op-amps. These may be filled in later without disturbing the design.



In a layout system, such freedom does not exist. The outside view of a device must resemble the internal view in its size and the location of connections. The designer cannot wire up an inverter, for example, without knowing the load the inverter is to drive, and hence its size. If at some later date a larger, more powerful inverter must be substituted, then the wiring must be changed. This additional complexity does not plague the designer who works with schematics.

Schematics also allow several shortcuts for indicating connectivity that have no equivalent in the world of IC design. Global signals may be defined whose values are known to all cells. Within a diagram, pins may be connected together simply by giving them the same name. In a layout, on the other hand, all connections of any type must be shown explicitly.

#### AN IMPLEMENTATION

A program has been developed to test this methodology. Component extraction, boundaries, assertions, methodology enforcement, and programmatic access have been implemented, but DRC has not. The system is written in PASCAL and runs on a DEC-20 with an HP-2648 graphics terminal and a Summagraphics BitPad One graphics tablet. PASCAL is also used as the imbedded language in which the user manipulates graphic constructs.

The program consists of two major sections, the editor and the compiler. The editor is the user interface, and allows the user to create, modify, and purge cells. Commands are similar to many other graphics editors, allowing the user to add components, interconnect them, move them and so forth. A minimal amount of checking is done at entry time for obvious errors, such as illegal signal names and lines at angles that the rest of the process cannot accept.

The editor and the compiler are adapted to various processes by means of a "process file". This file contains data on all the layers to be used in the compilation process. It is similar to the file normally used to describe DRC operations, with information about each layer, such as minimum spacing and width, capacitance per unit area, and the logical operations by which it is computed if it is not entered by the user.

The compiler is called whenever the user writes a cell out to permanent storage, thus making the cell available for use elsewhere. The compiler takes the user specifications of the geometry, fills out lines to their minimum width, and converts all input to polygons. The new layers are then computed, if necessary, and a connectivity analysis made. The DRC test is done at this point, so it can take advantage of the continuity information if it so desires. Then the boundary is computed (or the user specified boundary is checked) and every signal that touches the boundary is noted. The outline is then saved away so that it may be used by other cells. The date when the last change was made is recorded, so that when a cell is changed the modifications will propagate correctly.

The component extraction routines check for the obvious problems of multiply named signals, and signals of the same name that are not connected together. Running capacitance calculations are available for applications where performance is crucial. The user can ask for continuity to be shown; the system can take a signal by name or location and display on the screen where the signal is connected. All of the masks that have been generated as part of the DRC or component extraction process may be displayed.

Boundaries for cells may be specified in one of three ways. If nothing else is specified, then the boundary is computed as the logical OR of all the layers of the cell. Active areas are expanded by one minimum unit before the OR to ensure that no active area hits the boundary. This gives the minimum possible size for the cell, but the boundary is often far more complicated than necessary, and presents a cluttered appearance. It is never wrong, however, and is the default when data is transferred from another system that does not use the same methodology.

Boundaries may also be the minimum bounding boxes. This is consistent with several other artwork systems, and works well when the resulting cells are to be fed to an auto-router (since current auto-routers cannot handle arbitrary shape cells well). Finally, the user can specify the boundary by entering it as a polygon on a reserved mask. This usually yields the best results, but requires the most work.

Assertions appear to the user simply as notes with a reserved first character. Like signal names, they attach to the closest object. The compiler makes no attempt to process the assertions, but records them, along with the signals they are attached to, in the dictionary of signal names. This dictionary is available as a file, and any checking program that desires to do so may check it, and use the assertions as it sees fit. This is desirable since people will doubtless think up new ways to use assertions. In MOS design, for example, it may be necessary to include assertions about capacitances to be driven and voltage levels of various signals, as well as assertions about the signal timing.

The user tells the compiler to include a program along with a diagram by dividing the diagrams into FRAMES. Each frame has a name, and may contain primitives or text representing a program. If frames are present, and one is labelled PROGRAM, then the program in that frame is executed before any other frames are evaluated. The program may direct the evaluation of other frames, add graphics primitives to a diagram, and make decisions about the parameters of diagrams.

There are several restrictions imposed by this implementation. Although component extraction has been implemented for MOS transistors and parasitic capacitances, no DRC has been implemented yet. This should be a relatively simple project, since it only involves running a nearly conventional DRC on portions of the artwork that are selected by the compiler. Lines at angles other than 0 and 90 degrees are not supported, but this should be a straightforward extension. Although the simple logical operations such as AND, OR, and ANDNOT are implemented, the more complicated operations of OVERSIZE and UNDERSIZE are only implemented in a form that is not always

correct at the cell boundaries. If a cell is used in different surroundings, the user might prefer to create additional cells in order to do the geometrical operations properly and keep the results in the strictly no overlap format. At present this option is not available.

#### A POSSIBLE EXTENSION

An extension that allows more designer freedom at the cost of additional complexity has been proposed by Horowitz [Ho80]. This is to store a boundary for each layer instead of one boundary for the whole cell. This allows cells with internal terminals to be routed, and routing to be run over cells that were not specifically designed for this type of routing, if there is room. However, care must be exercised or else unintentional devices may be formed. Capacitance calculations and DRC checking also become more difficult, but the basic hierarchical structure of the solutions remain unchanged.

#### ACKNOWLEDGEMENTS

I would like to thank the design aids group at Hewlett-Packard for supporting this work, and my colleagues at Stanford University for many fruitful discussions. Martin Newell suggested including the analogy to structured programming.

#### REFERENCES

- [Pr79] Preas, Bryan T., "Placement and Routing Algorithms for Hierarchical Integrated Circuit Design", Technical Report #180, Computer Systems Lab, Stanford University, Stanford, CA. (August 1979)
- [McG80] McGrath, Edward J, and Whitney, Telle, "Design Integrity and Immunity Checking", Proceedings of the Seventeenth Design Automation Conference, Minneapolis, Minnesota, June 1980
- [Wh80] Whitney, Telle, "Description of the Hierarchical Design Rule Filter", SSP File #4027, Silicon Structures Project, California Institute of Technology, Pasadena CA. (October 1980)
- [McW80] McWilliams, Thomas M., "Verification of Timing Constraints on Large Digital Systems", Proceedings of the Seventeenth Design Automation Conference, Minneapolis, Minnesota, June 1980
- [Ho80] Horowitz, Mark. Private communication
- [Pe81] Penfield, Paul, and Rubinstein, Jorge, "Signal Delay in RC Networks", Caltech Conference on Very Large Scale Integration, January 1981.



*INNOVATIVE CIRCUIT DESIGNS SESSION*

*Chairperson: THOMAS F. KNIGHT, JR.  
Graduate Student  
Department of Electrical  
Stanford University*



# CONSIDERATIONS FOR AN ANALOG FOUR QUADRANT SC MULTIPLIER

by

Phillip E. Allen

and

William H. Cantrell

Department of Electrical Engineering

Texas A&M University

College Station, TX 77843

## ABSTRACT

This paper outlines the considerations and design of a four quadrant analog multiplier using switched capacitor (SC) techniques. The design algorithm for accomplishing the multiplication is described. Implementation of the algorithm is then presented. The predicted accuracy of the multiplier is given and compared to preliminary bread-board measurements. The multiplier described is presently being fabricated as an integrated circuit on a university multichip project using double-poly MOS technology.

## INTRODUCTION

Practical analog signal processing using integrated circuit technology has been made possible by the application of SC techniques.<sup>1</sup> The accuracy of analog signal processing systems can approach 0.1% which is equivalent to approximately 10 bits of digital information.<sup>2</sup> The primary analog signal function which has been implemented using SC techniques is the filter.<sup>3</sup> Some non-filter applications have also been considered such as oscillators,<sup>4</sup> diodes,<sup>5</sup> and digital-to-analog converters.<sup>6-9</sup> Some work has been done in the area of modulators,<sup>10</sup> but little if any consideration has been given to a true analog multiplier.

The objective of this paper is to take the proven techniques of SC circuits and apply them to the development of a four-quadrant SC multiplier. The result is a very useful analog signal processing component which is compatible with MOS technology. The speed of such a multiplier would not be expected to match the integrated bipolar analog four-quadrant multipliers presently available<sup>11</sup> because of the use of sampled data techniques. Preliminary results show that it is possible to eliminate many of the multiplier errors and to avoid the extensive fine tuning and external components that must accompany the use of bipolar analog four-quadrant multipliers. One useful technique that can be accomplished in SC circuits is to sample the offsets and cancel their contribution during the next clock phase period. This technique is considered as a means to reduce the errors associated with the analog, SC multiplier.

The paper will first consider the principles of operation by which the SC multiplier will be designed. These principles of operation will then be used to develop an implementation of the multiplier. An analysis of this implementation will provide the predicted performance which will be compared to breadboard results. This will be followed by the present status of this development and the future steps that will be taken. Brief consideration will be given to the implementation of



this multiplier using MOS technology which is presently under construction.

### PRINCIPLE OF OPERATION

In seeking a multiplier compatible with MOS technology, one might ask why not simply replace the bipolar junction transistors in the bipolar multiplier with MOS transistors. In theory, this approach proposed by the question should work. However, in practice there are several problems. The bipolar multiplier works on the principle of current ratioing using the transconductance of the bipolar junction transistor. Unfortunately, the MOS transistor has much lower transconductance and larger offset voltages leading to a four-quadrant multiplier having large errors. While the transconductance of the MOS transistors could be increased by using very large devices, the offset voltages would create too much error. With the concepts of SC circuits in mind, a new approach was sought which would take advantage of the SC methods to provide improved performance.

The operating principle chosen for the MOS multiplier can be explained by the block diagram in Fig. 1. This block diagram is used to represent the multiplier which will be designated as the operational multiplier. The operational multiplier has three inputs and one output. Two of the inputs are designated as multiplicands and the third input is called the divisor. The principle of operation can be simply stated as follows. Operate simultaneously on one of the multiplicands and the divisor in such a manner that the divisor is equal to the remaining multiplicand times a constant. For example, suppose that the operated value of the divisor  $C$  is equal to the multiplicand  $B$  and is given as

$$\text{Operated value of } C = KC = B \quad (1)$$

If the remaining multiplicand,  $A$ , is operated on in the same manner as

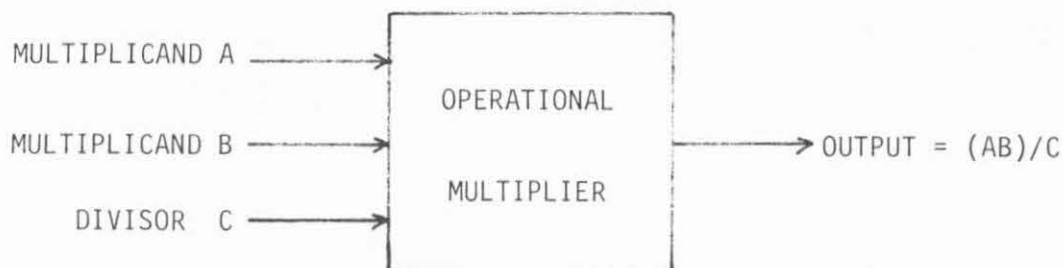


Fig. 1 - Block diagram of an operational multiplier.

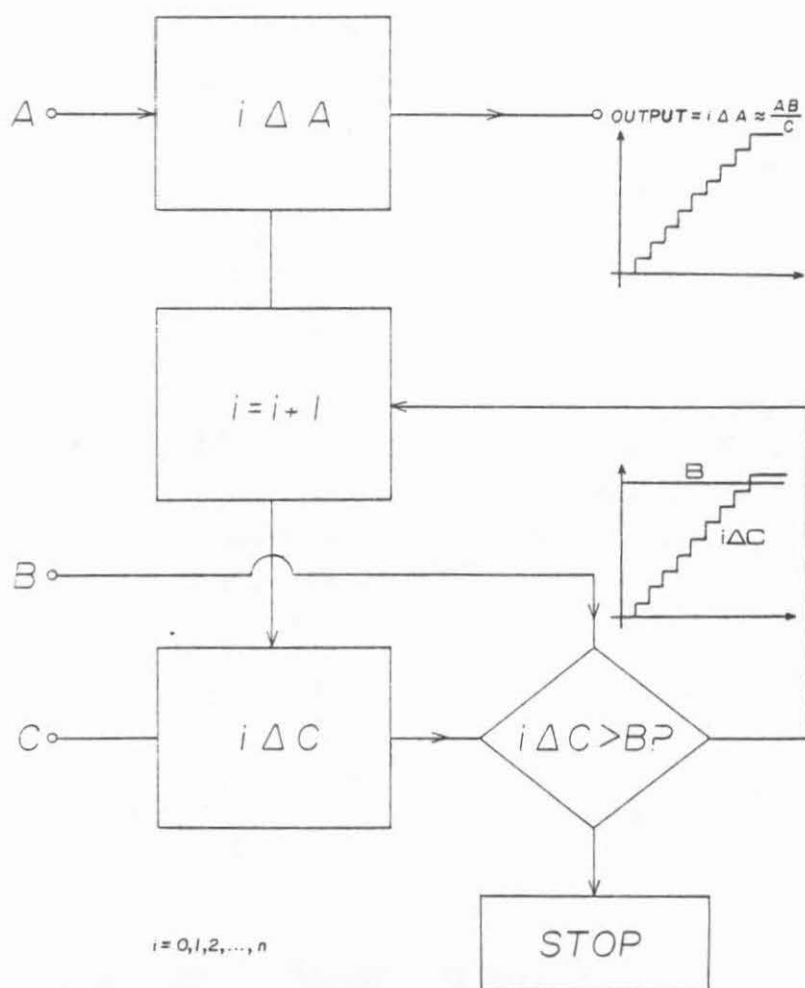


Fig. 2 - Counter implementation of the operational multiplier.

the divisor C then we can write

$$\text{Operated value of A} = KA \quad (2)$$

From (1) we see that  $K = B/C$ . If the output is equal to the operated value of A then the output can be expressed as

$$\text{Output} = \text{Operated value of A} = (AB)/C \quad (3)$$

If the output of the operational multiplier is equal to the operated value of A, then multiplication between the inputs A and B has been achieved. We also note that division of the product (AB) by C has also been achieved. Although the operations indicated above can be more complex than multiplying by a constant, this choice has the advantage of simplicity and is used in this paper.

### IMPLEMENTATION

The first implementation for analog multiplication to be considered was the counter approach as shown in Fig. 2. This approach is an obvious implementation of the operational multiplier. The counter approach consists of two accumulators designated as  $i\Delta A$  and  $i\Delta C$ . These accumulators continue to add  $\Delta A$  and  $\Delta C$  until  $i\Delta C$  is larger than B.

Once the accumulation stops, it is seen that B is equal to  $n \Delta C$ , or at least within one unit of  $\Delta$ . Obviously one can see that in order to achieve high accuracy the incremental constant,  $\Delta$ , must be very small. The problem arises in that as  $\Delta$  decreases, the number of steps,  $n$ , increases. If B is much greater than  $\Delta \cdot C$ , then a long interval will be necessary to obtain the output signal. The disadvantage of this approach is that the operation can take too much time, particularly if B is much greater than  $\Delta C$ .

The second method selected is the successive approximation approach as used in analog-to-digital converters, and is shown in Fig. 3. One can readily see the advantages of this approach over the previous one.

The master accumulator successively approximates the value of  $V_B$  resulting after  $n$  steps in

$$\sum_{i=0}^n \frac{b_i}{2^i} V_C \approx V_B \quad (4)$$

Rearranging terms:

$$\sum_{i=0}^n \frac{b_i}{2^i} \approx \frac{V_B}{V_C} \quad (5)$$

But,

$$V_{out} = \sum_{i=0}^n \frac{b_i}{2^i} V_A \quad (6)$$

Substituting,

$$V_{out} \approx \left( \frac{V_B}{V_C} \right) V_A \approx \frac{V_A \cdot V_B}{V_C} \quad (7)$$

Eq. (7) is approximate to within  $2^{-n}$  times  $V_C$ . This difference between eq. (3) and eq. (1) can be reduced by increasing the value of  $n$ . One can see that the successive approximation approach converges to the proper value much faster than the counter approach.

It turns out that this method is naturally adaptable to SC circuits and requires only three matched capacitors to implement the basic accumulator operation. Fig. 4 shows a circuit which resembles a SC integrator but has been modified for our purposes in implementing eq. (3). The circuit works as follows. At the beginning of the multiplying operation, the left-hand capacitor  $C$  is charged to the voltage  $V_C$ . The right-hand capacitor  $C$  is completely uncharged during this time. The multiplication operation begins by switching across both capacitors by exactly one-half. The value of  $V_C/2$  is then applied either positively or negatively to the capacitor  $C$  connected around

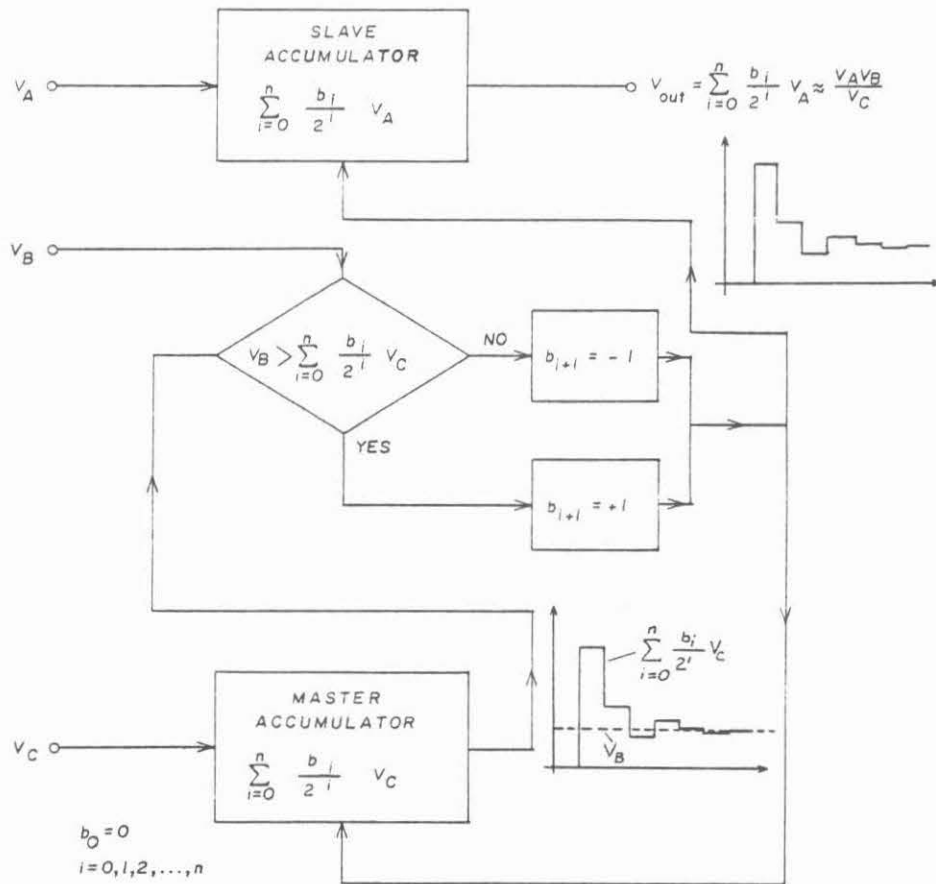


Fig. 3 - Successive approximation implementation of the operational multiplier.

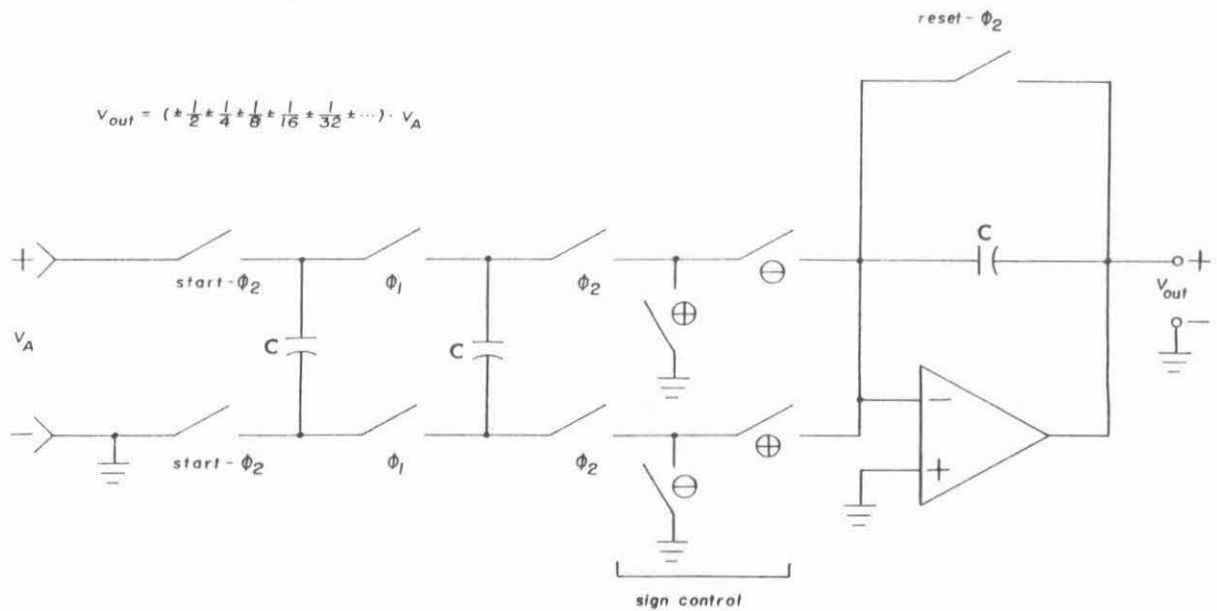


Fig. 4 - Switched capacitor accumulator circuit.

the op amp. This capacitor serves the purpose of a memory or an accumulator. The decision to accumulate in a positive or negative manner is made by comparing the output of the master accumulator with a reference voltage. If the accumulator is below this voltage, then the next sample is added to the accumulator. In this manner, the accumulator successively builds in value  $KV_C$  which approaches the reference voltage. A second accumulator operates on  $V_A$  in the same manner as the first accumulator resulting in the implementation of eq. (3). Since the accuracy of the accumulators is dependent upon how well the three capacitors designated as  $C$  can be matched, each sample of  $V_A$  should be transferred to the accumulator with an error of less than 0.1%.

Fig. 5 shows the implementation of the four-quadrant multiplier using the successive approximation accumulator of Fig. 4. The accumulators use a set of switches indicated as "+" and "-" to determine the polarity of the accumulation. In the "+" position the accumulator has the advantage of operating in a stray-sensitive mode which prevents capacitor and switch parasitics from affecting the accumulation.<sup>12</sup> In the negative position the accumulator is unfortunately susceptible to these parasitics which must be taken into account when considering the accuracy of the circuit. Note that the sign of the four-quadrant multiplier is automatically determined in Fig. 5. This is possible because the accumulators are bidirectional.

A shift register is used to control the operation and sequencing of the multiplier. The first output of the shift register is used to reset the accumulators by discharging the accumulation capacitance, and by charging the left-hand capacitor  $C$  to the value of  $V_A$  (or  $V_C$ ). The accumulation process continues until the shift register reaches the point where the sample-and-hold circuits are triggered and  $C$  is reconnected to the voltage  $V_A$  (or  $V_C$ ). The theoretical accuracy of the multiplier will be determined by the number of steps taken in the successive approximation sequence. Obviously, if no other considerations are pertinent, the accuracy times speed of performing a

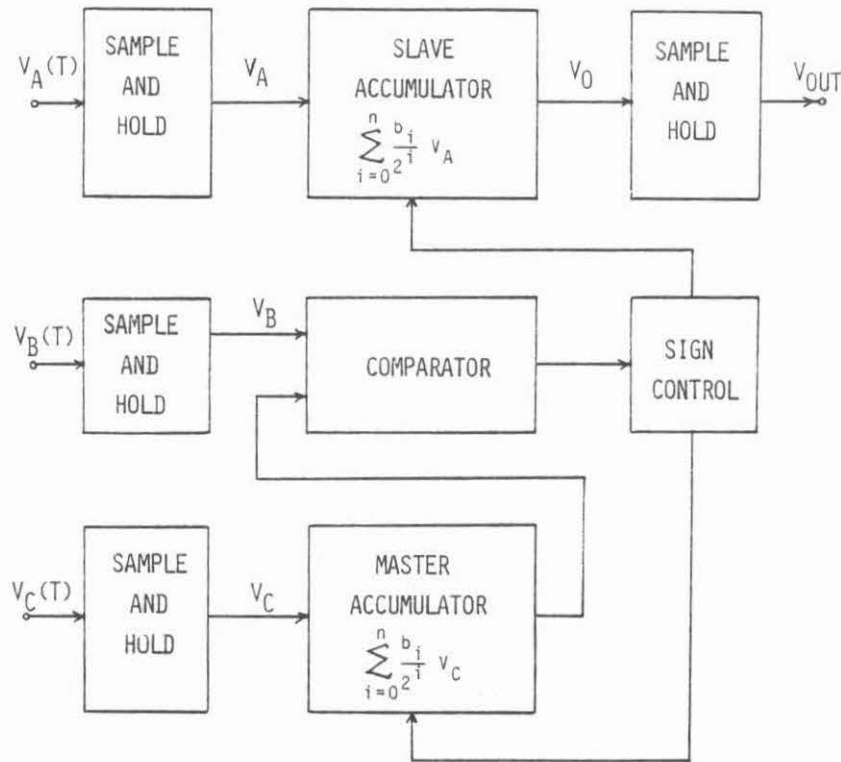


Fig. 5 - Block diagram of the successive approximation implementation.

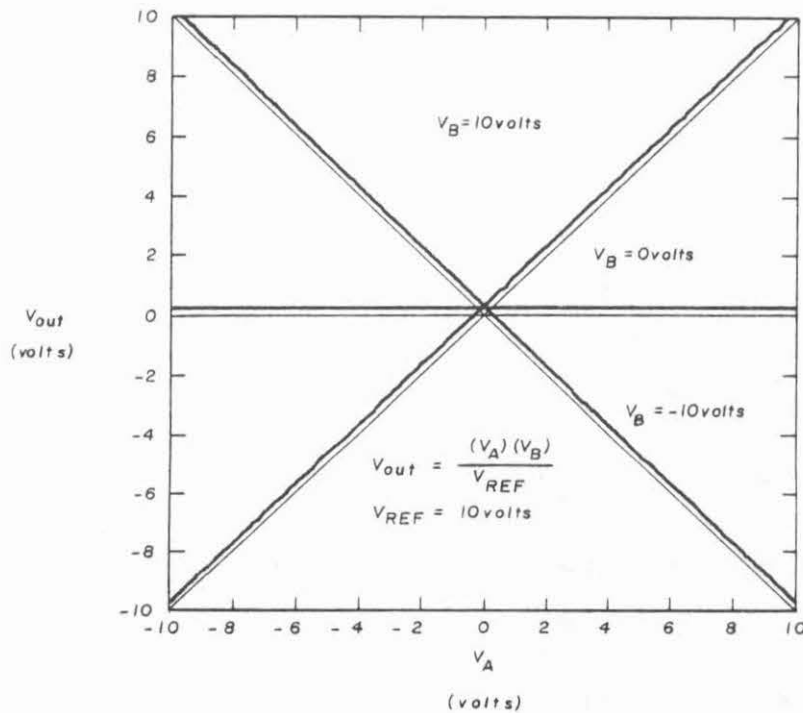


Fig. 6 - Experimental static characteristics of the multiplier.  $V_{OUT}$  versus  $V_A$  with  $V_B$  constant.

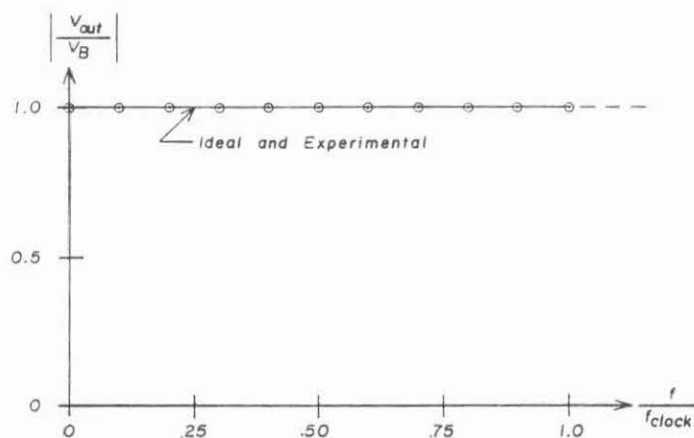
multiply operation would be a constant.

### PERFORMANCE

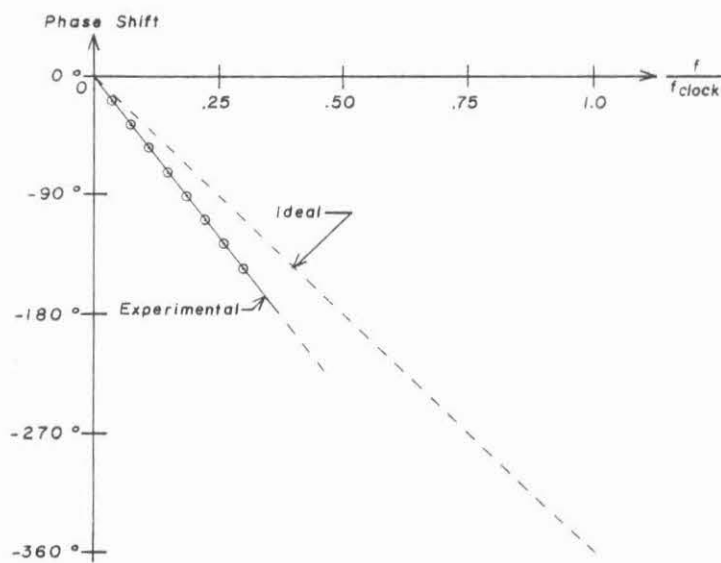
The static performance of a multiplier is typically characterized by offsets, feedthrough, and nonlinearities. Offset is due to an output voltage when both inputs are zero. Feedthrough is defined as an output when one of the inputs is zero and the other varies over its range of possible values. Nonlinearity has to do with the fact that the output of the multiplier may not be exactly equal to the product of the two inputs. The static performance of the breadboard version of the SC multiplier is shown in Fig. 6. To obtain these plots,  $V_C$  was connected to a +10 volt supply.  $V_A$  was set to -10, zero, and +10 volts to produce the three lines shown, and  $V_B$  was then swept over the range of -10 to +10 volts. As one can see, errors are mainly due to a constant offset of about 156 millivolts. This is a result of the number of successive approximations chosen for the breadboard version. Eight approximations are performed for each iteration. Since one approximation is discarded for resetting accumulators, seven useful approximations ( $n = 7$ ) remain. If in this case,  $V_C = 10$  volts, then the useful range of  $V$  is from -10 to +10 volts or 20 volts to  $2V_C$ . This means that the maximum error would then be  $2V_C / 2^n$  or  $20 / 2^7$  volts. Because this error is constant as shown, it can easily be nulled out by a single adjustment to be mentioned later. The results are similar when  $V_A$  is held constant and  $V_B$  is swept over its useful range.

In addition to these static characteristics, the bandwidth of the multiplier is also of interest. The bandwidth can be defined in terms of the magnitude response or the phase response. The best dynamic definition is the frequency at which a 1% absolute error is introduced in the multiplication operation. Figure 7 shows a plot of magnitude and phase versus frequency for a sine wave applied to the  $V_B$  input. The inputs  $V_A$  and  $V_C$  are each held at 10 volts. Note that the magni-





(a.) MAGNITUDE RESPONSE



(b.) PHASE RESPONSE

Fig. 7 - Frequency response of the SC multiplier. (a.) Magnitude and (b.) phase response.

tude remains constant regardless of frequency, and that the phase shift is a linear function of input frequency.

In the MOS version the offset error in Fig. 5 will become more important because of the clock feedthrough of the switches (not to be confused with the multiplier feedthrough). Because of the large clock signals and the possibility of high impedance states, small portions of the clock transitions will appear on the capacitors of Fig. 4. Although this feedthrough can become dependent upon the signal level, for purposes of this paper we shall assume that it is constant. Fortunately, in SC circuits we have the opportunity to sample the output offset and to introduce a cancelling component during the clock phases. To see how we can apply this idea to cancel the offset, let us consider the influence of the clock feedthrough. If the clock feedthrough introduces a constant component in the output, say  $\epsilon$ , then we can write the output of the two accumulators at the end of a multiplication sequence as

$$V_{\text{out}} = K V_A + \epsilon \quad (8)$$

and

$$V_B = K V_C + \epsilon \quad (9)$$

where  $K$  is expressed as

$$K = \pm \frac{1}{2} \pm \frac{1}{4} \pm \frac{1}{8} \dots \pm \frac{1}{2^n} \quad (10)$$

where  $n$  is the number of steps in the successive approximation of  $V_B$  of the multiplier. The approach taken to reduce this clock feedthrough is to build the dummy accumulator shown in Fig. 8. The dummy accumulator is identical to the other accumulators except that it has no input, and is not allowed to accumulate due to the discharge of the feedback capacitor  $C$  around the op amp during each clock cycle (this switch on the other two accumulators only operates once during the entire multiply sequence). In this manner a voltage will appear across

the two output capacitors which is caused by the clock feedthrough and the op amp offsets. These two capacitors will be applied on the same clock phase to the two accumulators in such a manner as to cancel the offsets due to the clock feedthrough and op amp offsets of these accumulators. If the dummy accumulator is matched to the actual accumulators, then the offsets should cancel. Furthermore, this system has the ability to track changes in the offset due to different clock amplitudes or temperature changes.

The nature of the operation of this multiplier prevents serious problems in multiplier feedthrough. If the A input is zero, then only clock feedthrough and op amp offset can contribute to an output but the dummy accumulator scheme of Fig. 8 should remove this output to minimize the B feedthrough. If the B input is zero,  $a_i$  should become close to zero since the dummy accumulator is removing the  $\epsilon$  value in eq. (3).

Since the error caused by mismatching of the capacitors is constant and since we are assuming that the clock feedthrough is not a function of the input amplitudes, then the nonlinearity of the multiplier should be very small. The dynamic range of the multiplier should be limited only by the power supplies and the ability of the dummy accumulator scheme to cancel the offsets caused by clock feedthrough and op amp offsets. Indeed, breadboard results of the SC multiplier circuit show that non-linearities are almost negligible with matched accumulators. More extensive measurements are being conducted at present.

## CONCLUSIONS

This paper considered the design of a SC multiplier which can be implemented using MOS technology. This circuit is compatible with other signal processing circuits designed by SC methods. The multiplier has four quadrant capability and has the potential of requiring no adjustments before application. The accuracy of the multiplier appears to be very comparable to existing multipliers and the opportunity to use offset cancelling techniques give the promise of excellent static characteristics. The circuit requires 6 op amps and 11 capaci-

tors in its present form including the dummy accumulator. At present, portions of Fig. 5 are being implemented using MOS technology to further study the effects of clock feedthrough and other sources of error to the multiplier operation. The next step will be to integrate the entire circuit and to analyze the performance of the system and to study potential applications.

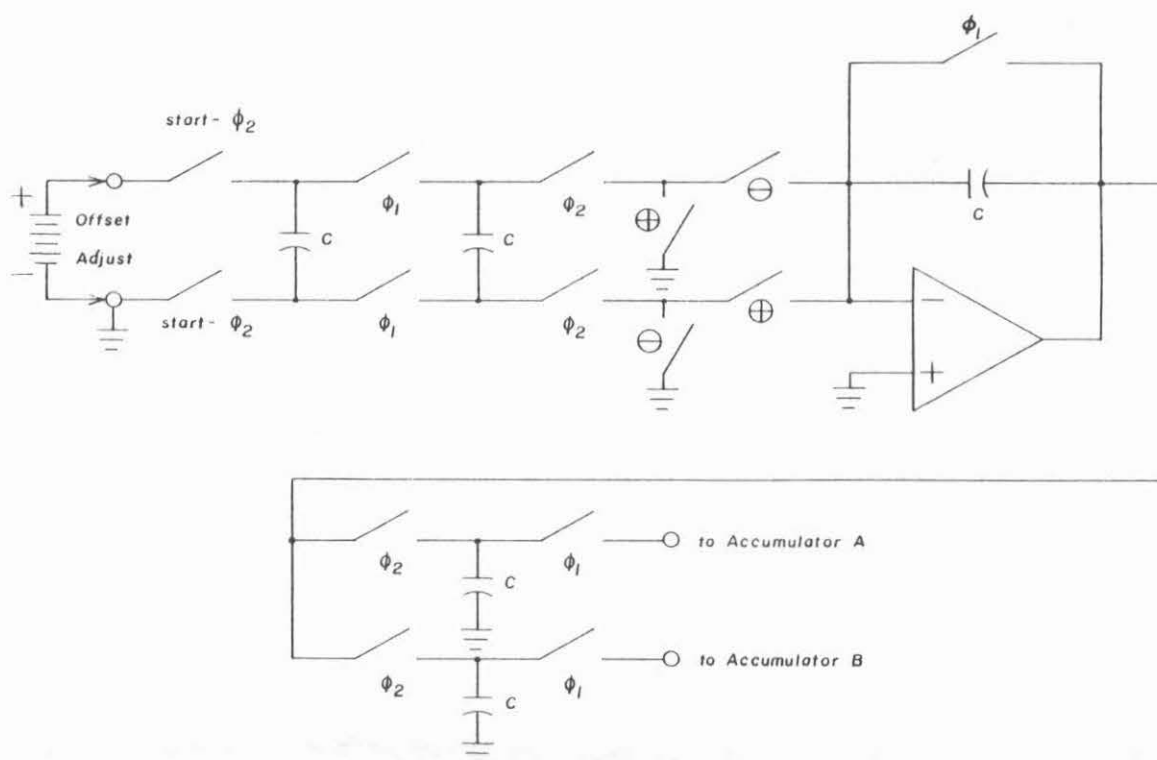


Fig. 8 - Dummy accumulator used to cancel offset.

## REFERENCES

1. D.A. Hodges, P.R. Gray, and R.W. Brodersen, "Potential of MOS Technologies for Analog Integrated Circuits", IEEE J. of Solid-State Circuits, Vol. SC-13, No. 3, June 1978, pp. 285-294.
2. J.L. McCreary and P.R. Gray, "All-MOS Charge Redistribution Analog-to-Digital Conversion Techniques - Part I", IEEE J. of Solid-State Circuits, Vol. SC-10, No. 6, pp. 371-379, Dec. 1975.
3. R.W. Brodersen, P.R. Gray, and D.A. Hodges, "MOS Switched-Capacitor Filters", Proceedings of the IEEE, Vol. 67, No. 1, pp. 61-75, Jan. 1979.
4. T.R. Viswanathan, K. Singhal, and G. Metzker, "Application of Switched Capacitor Resistors in RC Oscillators", Electronic Letters, Sept. 28, 1978, Vol. 14, No. 20, pp. 659-660.
5. B.J. Hosticka and G.S. Moschytz, "Practical Design of Switched-Capacitor Networks for Integrated Circuit Implementation", Electronic Circuits and Systems, March 1979, Vol. 3, No. 2, pp. 76-88.
6. Y.S. Lee, L.M. Terman and L.G. Heller, "A Two-Stage Weighted Capacitor Network for D/A - A/D Conversion", IEEE J. of Solid-State Circuits, Vol. SC-14, No. 4, August 1979, pp. 778-781.
7. K.B. Ohri and M.J. Callahan, Jr., "Integrated PCM Coder", IEEE J. of Solid-State Circuits, Vol. SC-14, No. 1, Feb. 1979, pp. 38-46.
8. A.R. Hamade, "A Single Chip All-MOS 8-Bit A/D Converter", IEEE J. of Solid-State Circuits, Vol. SC-13, No. 6, Dec. 1978, pp. 785-791.
9. B. Fotouki and D.A. Hodges, "High-Resolution A/D Conversion in MOS/LSI", IEEE J. of Solid-State Circuits, Vol. SC-14, No. 6, Dec. 1979, pp. 920-926.
10. J. Bingham, "SC Modulators", Switched-Capacitor Workshop, Feb. 10-12, 1980, Palo Alto, CA.
11. B.A. Gilbert, "A Precise Four-Quadrant Multiplier with Subnanosecond Response", IEEE Journal of Solid-State Circuits, Vol. SC-3, No. 4, Dec. 1968, pp. 365-373.
12. K. Martin and A.S. Sedra, "Stray-insensitive Switched-capacitor Filters Based on Bilinear Z-Transform", Electronics Letters, Vol. 15, No. 13, June 21, 1979, pp. 365-366.



# A One Transistor RAM for MPC Projects

by

James J. Cherry and Gerald L. Roylance

Massachusetts Institute of Technology

545 Technology Square, Cambridge, MA

**Abstract:** Many MPC projects, such as video frame buffers, need a large memory subsystem. A one transistor per bit dynamic memory using Mead-Conway design rules is being designed with this purpose in mind. The memory cell size is  $16.5\lambda$  by  $8\lambda$  (about the same size as a 1975 4K RAM cell with  $\lambda = 2.5$  microns).

While a complete high density memory subsystem has not been designed, two chips have been designed to test its major components. One chip is a 1K memory array that tests the sense amplifier, column decoder/driver, and read/write logic. This chip lacks a timing generator and clock drivers. The second chip tests some low power bootstrapped clock drivers. These test chips are currently being fabricated.

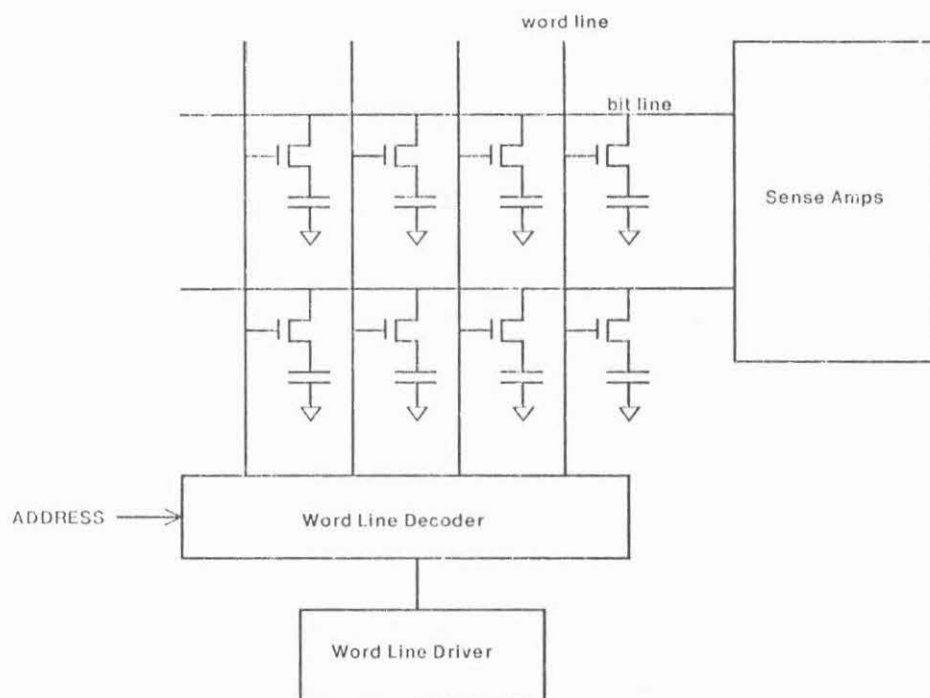
This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's V. L. S. I. research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research Contract number N00014-80-C-0622 and in part by the Advanced Research Projects Agency under Office of Naval Research contract N00014-75-C-0643.

## 1. Design Considerations for a One Transistor RAM

Many VLSI projects need a moderately large memory module. A one transistor dynamic memory has been designed as a subsystem usable in the Multi-Project Chip (MPC) designs being undertaken by several universities. Because the memory is intended for MPC projects, it follows the conservative Mead-Conway design rules and uses only a single layer of polysilicon; consequently, the achievable memory density suffers. Furthermore, the memory design must tolerate wide process variations because many different fabrication lines are used for MPC.

Throughout the RAM design we have decided in favor of simplicity in order to give the RAM the best chance of working. When the choice was between speed and density, we chose density because we feel density is more important to the average project.

A block diagram of the memory is shown in the first figure. The major components are the memory array, sense amplifiers, column address decoder, and word line (column) driver. The bits of the memory are stored as a voltage (0 or 2 volts in our case) on the capacitors in the array. All of the memory cells in a column are read at once. Addressing is done by the word line decoder; the decoder takes a positive pulse from the word line driver and steers that pulse to the column selected by the address lines. That word line pulse turns on all of the pass transistors in the selected column, thus connecting the memory cell capacitor to the horizontal bit lines and to the sense amplifiers where the binary value held in the cell is determined. The sense amplifier must be sensitive to signals on the order of a hundred millivolts because the memory cell capacitance and the (much larger) stray bit line capacitance form a voltage divider. In the present design this attenuation is a factor of 15.



There are several references on one transistor RAM design. Barnes [1] gives a short description of several sense amplifier designs. We used the charge transfer sense amplifier first reported by Heller [5, 6]. We did not use the more



sophisticated version of this amplifier described in Heller [7] because we felt there would be a better chance of making the simpler version work. The charge transfer sense amplifiers have more sensitivity than amplifiers that directly sense the voltage difference on the bit lines and are also tolerant of bit line capacitance and transistor threshold voltage variations.

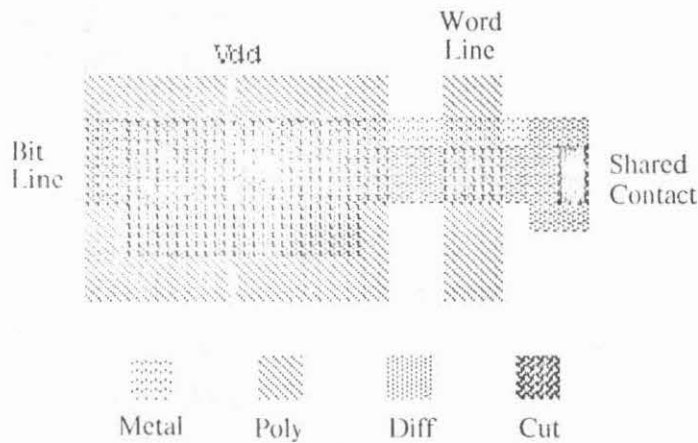
The input-output (I/O) circuits [not shown] follow those of Gray [4]. The I/O is done on one side of the array (rather than in the center of the the array) so many bits (for example, 32 bits) can be read or written at once, allowing higher memory bandwidth than that available from commercial parts that are limited to reading from 1 to 8 bits at a time.

The high voltage (7.5 volt) bootstrap driver used to precharge the bit lines is based on one described by Chan [2]. The word line decoder is derived from one given by Tzou in an article on CCD memories [11].

## 2. Memory Cell Design Considerations

Several different memory cell layouts were tried in the search for highest possible array density. The densest one we found (shown below) uses metal bit lines and polysilicon word lines. The memory capacitor is actually an enhancement mode transistor whose source and drain are tied together to make one terminal and whose gate forms the other terminal; this latter terminal is connected to  $V_{DD}$ .

For a bit line capacitance to storage capacitance ratio of 15 to 1 (with 64 cells on each bit line), the cell size is  $16.5 \lambda$  by  $8 \lambda$ . This cell is half the size (comparing square  $\lambda$ 's) of a three transistor RAM cell designed by Dick Lyon at Xerox PARC. With  $\lambda = 2.5 \mu$ , this cell is the same size as the INTEL 2104, a commercial dynamic RAM that came out in 1975.



Memory Cell Layout

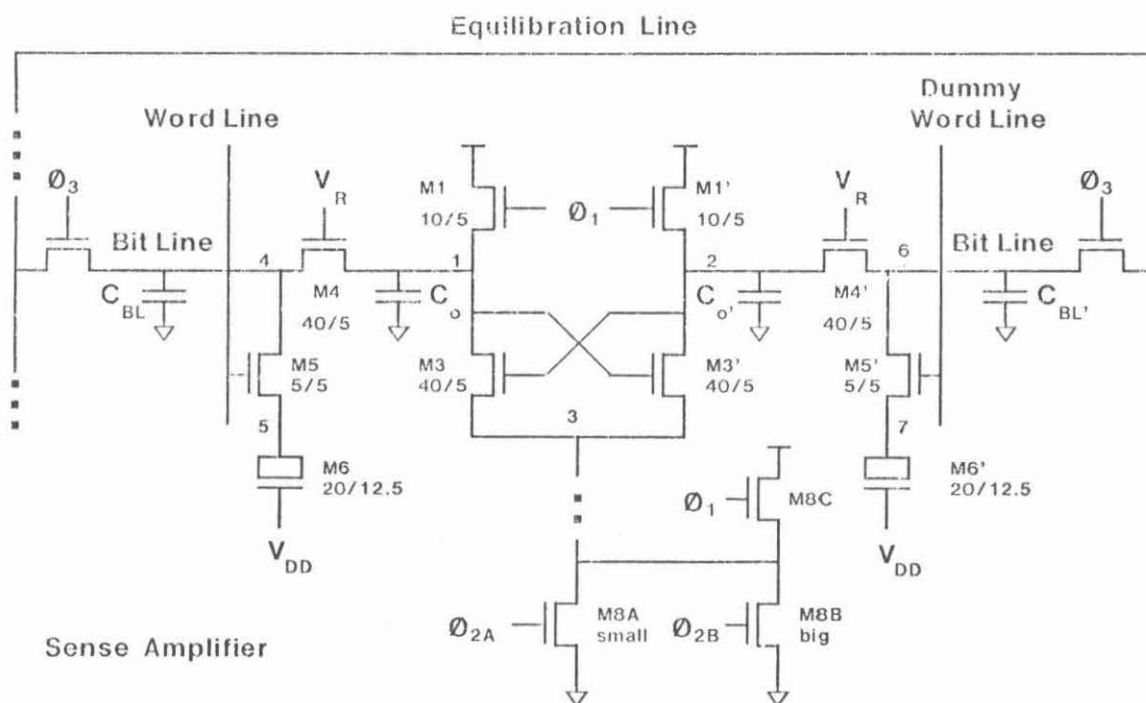
Cell size is not the only size consideration. The sense amplifiers and the word line decoders must also fit in the pitch determined by the cell size. While there was little problem with the decoder pitch, the tightest pitch we could give the sense amplifier was  $11\lambda$  (when we needed  $8\lambda$ ). Although a bit line pitch of  $11\lambda$  could be used, it would cause a significant increase in cell area. In a two level polysilicon process (which we do not have), the area penalty can be avoided by using a folded bit line [10] or a staggered bit line [8].

We were able to use the minimum area cell (with its  $8\lambda$  pitch) by placing two amplifiers side by side and fitting the pair to a  $16\lambda$  pitch. This layout is not symmetrical and some effort is needed to balance the stray coupling from control lines onto the bit lines or the memory will not work. The noise injected onto the bit lines by these control lines is one-half of the signal that the amplifier is trying to sense!

### 3. Sense Amplifier

The heart of every sense amplifier design is a cross-coupled differential pair (M3 - M3' in the figure below). A small initial voltage difference on nodes 1 and 2 is amplified when node 3 is pulled down by M8. The main difference between the many different sense amplifiers used in dynamic RAMs is how the cross coupled latch is connected to the bit lines [1]. In order to sense a small differential voltage quickly, the large capacitance of the bit lines must be isolated from the internal nodes of the differential amplifier.

The next figure shows the basic *charge transfer sense amplifier* that we use and that was first described by Heller [5, 6]. (Much of the peripheral circuitry that we use comes from Gray [4]).



The memory cell capacitors are FET's M6 and M6'.  $\phi_1$  is a high voltage pulse (above  $V_{DD}$ ) that precharges nodes 1, 2, and 3 to  $V_{DD}$ .  $V_R$  is a supply voltage less than  $V_{DD}$ , so nodes 4 and 6 charge to  $V_R - V_T$  ( $V_T$  is the transistor threshold voltage) as M4 and M4' reach cut-off. Since  $V_R - V_T$  is the highest the bit lines can

go, this voltage is used to store a logical *one* in the memory cells. A logical *zero* is represented by storing zero volts in the memory cell capacitor. A voltage half way between these two limits is stored on a dummy cell (M6') to provide a reference voltage for the sense amplifier. Each bit line is populated with 64 memory cells and one dummy cell. When reading a column of bits from one side of the array, the dummy cell on the opposite side of the array is addressed. Thus, the figure above depicts the the situation encountered when reading a bit out of the left half of the memory array.

When a read sequence starts, the precharge line ( $\varphi_1$ ) is turned off and the word line and dummy word lines are brought high to connect a memory cell (M6) and the dummy cell (M6') to opposite sides of the differential sense amplifier. Consider the case in which a zero is stored in M6. The **drop** in voltage on the bit line will be:

$$\Delta V_4 = (V_R - V_T) C_S / (C_S + C_{BL})$$

where  $C_S$  is the capacitance of the memory cell, and  $C_{BL}$  is the bit line capacitance. This takes M4 out of cut-off and into saturation with a gate drive of  $\Delta V_4$ . M4 will remain on until the bit line is charged to back to  $V_R - V_T$ . The charge necessary to charge  $C_S + C_{BL}$  back to  $V_R - V_T$  must come from  $C_o$ , the parasitic capacitance on node 1. Thus,

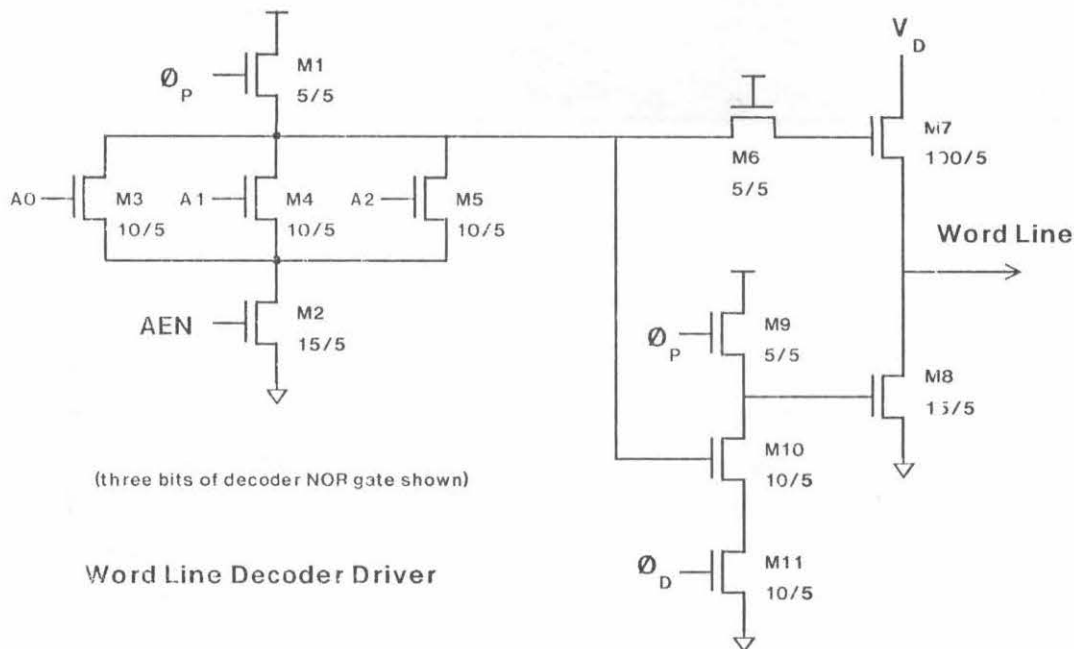
$$\Delta V_2 = (V_R - V_T) C_S / C_o$$

The voltage change at node 3 will be one half of this. Note that the differential voltage  $\Delta V_2 - \Delta V_3$  is independent of the bit line capacitance if enough time has elapsed [5, 6].  $\varphi_{2A}$  comes on and starts pulling node 3 toward ground, eventually turning one of M3 or M3' on. Node 3 is pulled down slowly enough to prevent the other latch transistor from turning on [9]. As the  $V_{GS}$  of the on transistor increases, so does it's  $g_m$ .  $\varphi_{2B}$  is then turned on to quickly pull node 2 to ground. The bit that was stored on M6 has now been refreshed, so the word line is turned off.  $\varphi_{2A}$  and  $\varphi_{2B}$  are turned off.  $\varphi_3$  is turned on to short **all** of the bit lines together; since half of the bit lines are at the logic zero level (0 volts) and the other half are at logic one ( $V_r - V_l$ ), the bit lines go to a voltage half way between the two levels. This voltage is stored in the dummy cell by dropping the dummy word line and  $\varphi_3$ .

A memory write cycle proceeds exactly like that of a read, but instead of letting the contents of the memory cell determine the fate of the bit line, a circuit at the end of the bit line either pulls it to ground or pushes it up with a capacitor approximately the same size as a memory cell [4].

#### **4. Word Line Decoder**

It is impractical to use a separate high capacitance driver for each of the 128 word lines, so a single driver must be shared by many word lines. The word line decoder does both the memory addressing and the multiplexing of the high capacitance driver. The word line decoder also clamps the unselected word lines to ground to minimize some problems related to subthreshold conduction of the FET's that isolate the unselected memory capacitors to the bit lines [2, 4]. The word line decoder circuit shown below is a modified version of the one described by Tzou [11]. The basic idea is to bootstrap pass transistor M7 with its own gate-source capacitance so that all of the voltage developed by the word line driver is delivered onto the word line.



Operation of the decoder driver starts with  $\phi_P$  precharging the gates of M8 (the clamp transistor) and M7 (the pass transistor) while  $V_D$  (word line driver),  $\phi_D$ , and AEN are all low. When the address lines (A0-A6) have settled,  $\phi_P$  goes low followed by AEN going high. If all of the address lines inputs of the NOR gate decoder are low, then the word line is selected and the pass transistor will allow the word line pulse to pass through. In that case, when  $\phi_D$  comes on it will discharge the gate of M8, allowing the output to rise when  $V_D$  comes along. As  $V_D$  rises, isolation transistor M6 turns off allowing M7 to bootstrap.

If one of the address lines in the NOR decoder is high, then the word line is not selected. When AEN goes high, the gate of M7 is discharged, turning it off and isolating the word line from the driver.  $\phi_D$  is prevented from discharging the gate of M8, the clamp transistor, by M10.

Tzou's design does not include isolation transistor M6. Without this transistor, much of the bootstrap charge on M7 is lost in charging the diffusion capacitance of the decoder logic. Another addition to Tzou's design is M2, which provides a simple means of disabling the address lines.

The capacitance associated with the drain of M7 is a surprisingly large load to the word line driver. While the word line capacitance for a 16Kbit version of the RAM for a typical process would be only 2pF, the combined drain capacitances of 128 decoders is several times higher (about 8pF). Our first layout of M7 used a straight gate and had a capacitance from the drain to ground of .1pF. Bending the gate into a rectangle (a suggestion of Pat Bosshart) eliminated the sidewall capacitance and reduced the capacitance to .068pF per drain. Another layout, suggested by Tom Knight but not used by us, has an effective capacitance of .056pF.

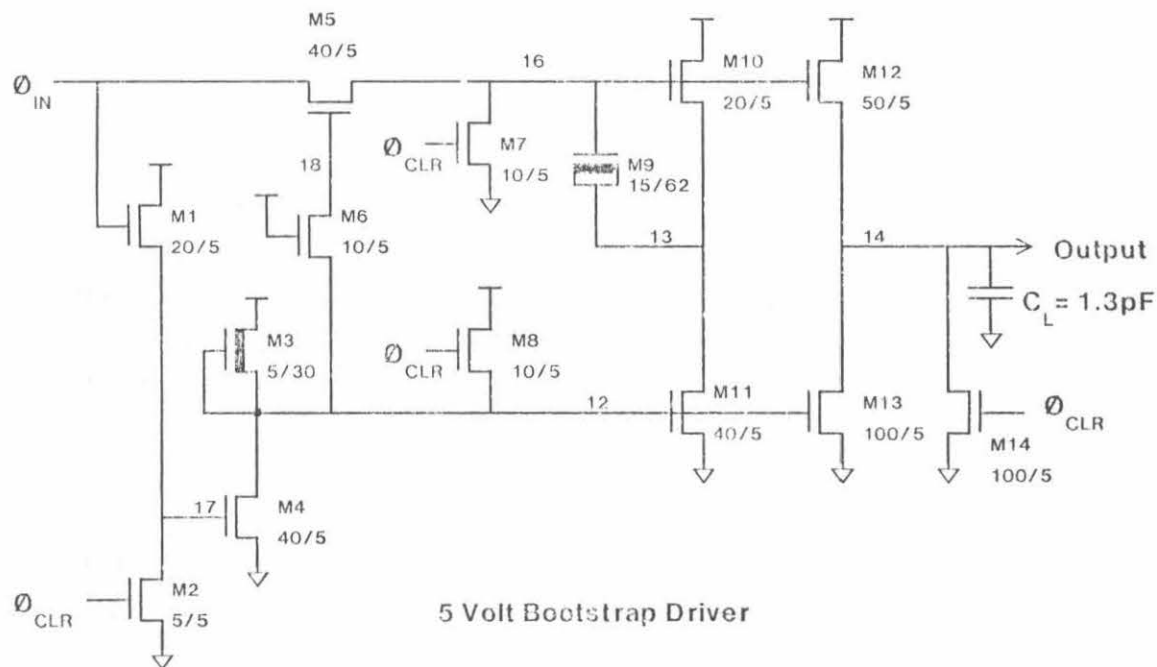
## 5. Bootstrap Drivers

In a typical application the memory array is large and consequently the clock signals are large capacitive loads which require driver circuits. We designed two bootstrap drivers: one is a 5 volt driver for the address inputs of the word line decoder and  $\varphi_3$  of the sense amplifier and the other is a 7.5 volt driver for precharging the bit lines ( $\varphi_1$ ). A third driver which is for the word lines ( $V_D$ ) has not been designed. The 5 volt circuit takes 5nS to switch a 1.3pF load (128  $5\mu \times 5\mu$  gates); the 7.5 volt circuit takes 15nS to drive the same load.

Neither of the driver circuits is connected to the memory array so that the drivers and the memory can be tested independently. Super buffers are used for the address buffers in the current memory array, but all other clocks are simply bonded out to pads. Normal depletion load inverters and super buffers are not acceptable for the word line driver because their logic zero output is not 0.0 volts. Super buffers consume more standby power than a dynamic bootstrap driver and so super buffers are less desirable in a large memory.

The 5 volt bootstrap driver is modeled after one used in the INTEL 2118 16K dynamic RAM (see figure).  $\varphi_{CL,R}$  precharges the gates of M11 and M13 high, turning them on. When  $\varphi_{IN}$  starts to rise, it charges capacitor M9 and starts to turn M10 and M12 on. M6 isolates node 18, allowing that node to bootstrap and keep M5 turned on hard. M1 and M4 form a comparator that notices when  $\varphi_{IN}$  has gone above 2 threshold drops. When this happens, M4 turns on and pulls nodes 12 and

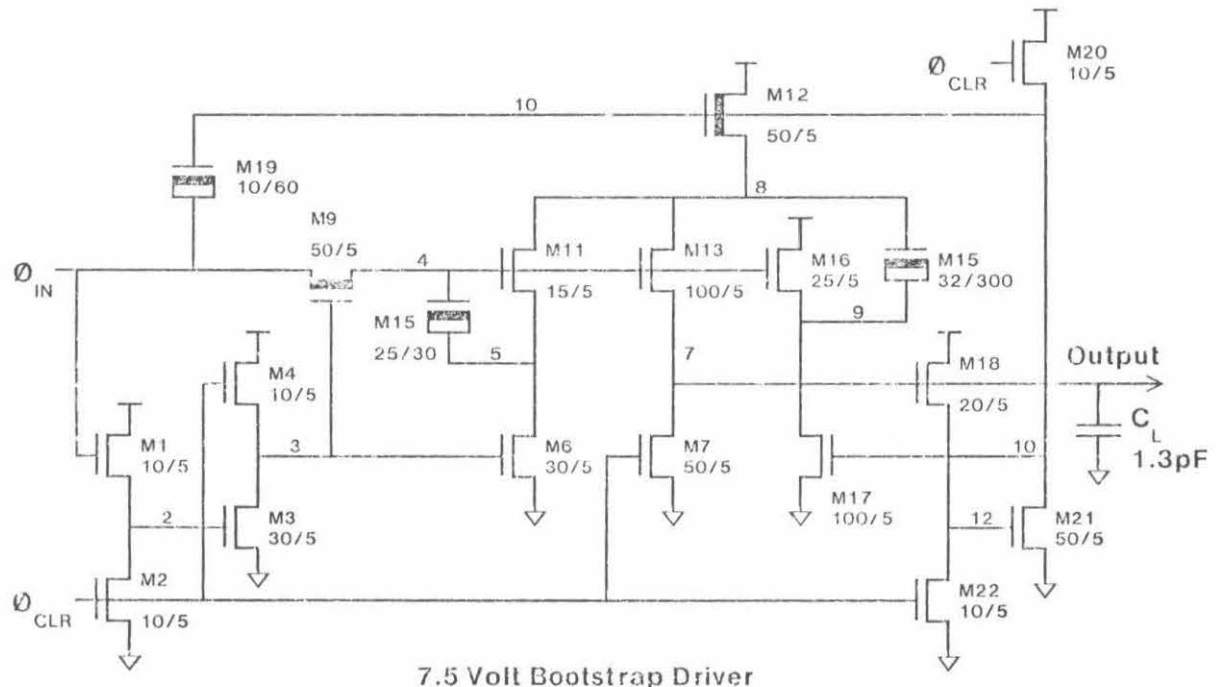
18 down to ground. M11 and M13, which had been holding down nodes 13 and 14, now turn off, letting those nodes rise. Capacitor M9 bootstraps node 16 (which was isolated by M5 when M5's gate fell), turning M10 and M12 on hard. M12 pulls the output node voltage up.  $\varphi_{IN}$  can now fall without affecting the rest of the circuit because M5 is off.  $\varphi_{CLK}$  turns M11 and M13 on and turns off M10 and M12, forcing the output low and resetting the circuit. The bootstrap capacitor M9 is driven from node 13 and not from node 14 to get more gate drive on M12 which significantly improves the output rise time.



The high voltage bootstrap driver (after Chan [2]) is basically two of the 5 volt drivers that have been merged together. M15 bootstraps in much the same way as M9 did in the previous circuit. When M15 is bootstrapping, node 10 (which was pushed high by  $\varphi_{IN}$ ) connects node 8 to  $V_{DD}$ . As node 4 rises, the output (node 7) also rises. M18 and M21 form another comparator that notices when the output has exceeded two  $V_T$ ; then M21 pulls down node 10, turning off M17 which had been holding one terminal of capacitor M15 down at ground and also turning off M12 which had been holding the other terminal at  $V_{DD}$ . M16 has been turned on because node 4 has been bootstrapped high earlier. M15 now bootstraps node 8.



M11 is still on, so node 5 follows node 8 which pushes node 4 still higher. Node 4 makes M13 and M16 stay turned on; thus, the output follows node 8. The clever feature of this circuit is that M15 does not charge share with the load capacitance until the load capacitance has been charged up to two  $V_T$  (ie, until the comparator trips).



## 6. Conclusion

We designed two project chips to test our designs. One chip is a 128 by 8 (1K) array of memory cells with sense amplifiers, word line decoders (implemented with super buffer address drivers), and multiplexed read/write logic. This array size provides a reasonable feasibility test for building a 16K subsystem. No clock or timing generation is included on the chip because we did not have enough time. We expect access times of 150ns and cycle times of 250ns. A second, separate project chip test the 5 volt and the 7.5 volt bootstrap drivers. Additional circuitry is included on that chip for measuring the capacitive loading on the drivers when a

low capacitance probe is connected to their outputs.

The original goal of this effort was to develop a high density memory subsystem that could be treated as a "black box" by designers with little or no analog background. We severely underestimated the magnitude of such a task. As it stands, our results can only be viewed as a first cut towards that goal. We found that many design challenges lie in the peripheral circuitry such as drivers and decoders: *there is much more to a one transistor RAM than sense amplifiers.*

We thank Mark Johnson for telling us about laying out high frequency gate oxide capacitors and for spotting the undesirable control line coupling in the sense amplifier. We thank Tom Knight for general guidance and moral support. The RAM was designed as a term project for an MOS analog circuit design course taught by Prof. Yannis Tsividis of Columbia University while visiting MIT. Prof. Tsividis has given both of us a better understanding of using MOSFETs in both analog and digital design.

## 7. Bibliography

1. J. Barnes, "A High Performance Sense Amplifier for a 5V Dynamic RAM", *IEEE Journal of Solid-State Circuits*, Vol SC-15, No. 5, October 1980, pp 831-839.
2. J. Y. Chan, et al, "A 100nS 5V Only 64Kx1 MOS Dynamic RAM", *IEEE Journal of Solid-State Circuits*, Vol SC-15, No. 5, October 1980, pp 839-846.
3. Electronic Design, "Circuit Techniques Tune Up for Production of 64-K RAMs", *Electronic Design*, October 25, 1980, pp 31-32.
4. K. Gray, "Cross-Coupled Charge-Transfer Sense Amplifier and Latch Sense Scheme for High-Density FET Memories", *IBM Journal of Research and Development*, Vol 24, No. 3, May 1980, pp 283-290.

5. L. G. Heller, D. P. Spampinato, Y. L. Yao, "High-Sensitivity Charge-Transfer Sense Amplifier", 1975 IEEE International Solid-State Circuits Conference, pp 112-113.
6. L. G. Heller, D. P. Spampinato, Y. L. Yao, "High Sensitivity Charge-Transfer Sense Amplifier", IEEE Journal of Solid-State Circuits, Vol. SC-11, No. 5, October 1975, pp 596-601.
7. L. G. Heller, "Cross-Coupled Charge-Transfer Sense Amplifier", 1979 IEEE International Solid-State Circuits Conference, Feb 1979, pp 20-21.
8. T. C. Lo, R. E. Scheuerlein, R. Tannlyn, "A 64K Dynamic Random Access Memory: Design considerations and Description", IBM Journal of Research and Development, Vol. 24, No. 3, May 1980, pp 318-327.
9. W. T. Lynch, H. J. Boll, "Optimization of the Latching Pulse for Dynamic Flip-Flop Sensors", IEEE Journal of Solid-State Circuits, Vol SC-9, No. 2, April 1974, pp 49-55.
10. F. J. Smith, R. T. Yu, I. Lee, S. S. Wong, M. P. Embrathury, "A 64 kbit MOS Dynamic RAM with Novel Memory Capacitor", IEEE Journal of Solid-State Circuits, Vol. SC-15, No. 2, April 1980, pp 184-189.
11. A. Tzou, et al, "A 256K-Bit Charge-Coupled Device Memory", IBM Journal of Research and Development, Vol 24, No. 3, May 1980, pp 328-338.
12. Y. S. Yee, L. M. Terman, L. G. Heller, "A 1 mV MOS Comparator", IEEE Journal of Solid-State Circuits, Vol. SC-13, No. 3, June 1978, pp 294-297.



## PLA Design in NAND Structure

Chong Ming Lin

Semiconductor Engineering Group

Digital Equipment Corporation  
75 Reed Road  
Hudson, MA 01749 (617) 568-4888

ABSTRACT--A NAND (serial gating) structure PLA of the MOS poly-silicon gate process has been developed for high density and medium fast speed VLSI application. Dynamic clocking is used for minimum power dissipation and elimination of the ratio problem associated with static NAND gate. Ion-implantation for memory cell programming and the elimination of contact in the memory area drastically reduces the cell size, and reliability is improved. A simple but effective self-timed clocking scheme is employed for better operating margins against process variations; the overhead chip area for the clock generation is sufficiently small. The advantages of allowing metal signal and power lines to cross the PLA memory area is discussed. Some measured data from a 3.5 $\mu$ m NMOS Si-gate process with regard to gate height and transistor sizes are also described.

## INTRODUCTION

In MOS circuit design, the NAND circuit, due to its inherited electrical characteristics, has been restricted in application for only a limited number of inputs. In the past, ion implant programmed NAND structures had only been used for very slow speed ROM applications [1] [2]. However, with the fast progress in process technology, a properly designed NAND structure PLA is becoming more attractive for some of the existing applications. New product of a new structure with a new process is always impressive for its performance, but its cost effectiveness is not guaranteed on production level. This question was better stated by G. Moore in his lecture, in the 1st Caltech VLSI Conference, January 1980:

"... the semiconductor industry is not now process--technology limited for non-memory product. How to best make use of the processing technology is really what the problem is."

## PLA Design in NAND Structure

The experiment discussed in this report was done primarily as an answer to a request, in November 1978, for a high density and medium fast speed PLA design. In order to make best use of a then newly developed 3.5  $\mu\text{m}$  process and fully utilize the given timing spec of the speed requirement, a NAND structure PLA was proposed for better performance which was also cost effective for an existing application and beyond.

In order to achieve overall low chip size and a high operating margin; a dynamic, self-timed clocking scheme was proposed for most of the circuitries. Knowledge from measured data are used to construct circuits with better performance than the original approaches used on a circuit test chip [3].

Since the NAND and the NOR structures are the two basic building blocks and complementary to each other in MOS circuit design, the author feels that the study is also useful for understanding the general NOR type PLA design as an expected by-product.

In the following sections, process background, NAND circuit model, cell layout, reliability and design consideration will be described. Further improvement will also be discussed.

### PROCESS BACKGROUND

Although the process performance is crucial in the evaluation of a new circuit structure, this information was not available in the previous papers [1] [2]. In order to test out some of the assumptions and limitations of the NAND structure PLA, a test chip was designed and manufactured in 1979 with a 3.5  $\mu\text{m}$  NMOS process, P400. This process uses ion implantation for Source/Drain, plasma dry etching, silicon doped aluminum; plus 4 types of devices, as shown in Table I, which provide more flexibility in circuit design and better powerXspeedXdensity product than many of the processes in the previous generation.

The performance of the process was measured through a ring-oscillator built on the test chip. As shown in Fig. 1(c), the powerXspeed product curves indicate that in a typical case, the performance of that ring-oscillator is around 0.35 pJ at 5V VCC, -3V VBB, and RT. Those curves also indicate that the process parameters are being optimized against those skewed process corners. With this kind of performance, the NAND PLA does have better potential for some applications which were not feasible by processes of the previous generations.

TABLE I

Device threshold voltages and ion-implantation definition of the P400 process.

	Boron	Arsenic	Vth (typical)
Enhancement	x	--	0.7v
Intrinsic	--	--	0.0v
Depletion-1	x	--	-1.2v
Depletion-2	--	x	-2.5v

Vert. 1v/Div. Horl. 10ns/Div.

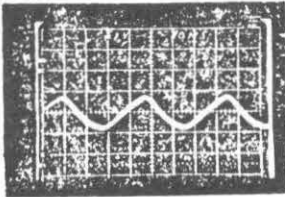


Fig. 1b- The wave form of the ring-oscillator, measured at 23 C, 5V VCC, and -3V VBB.

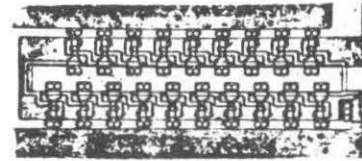


Fig. 1a - Photomicrograph of a ring-oscillator with 19 stages and fan-in and fan-out of 1.

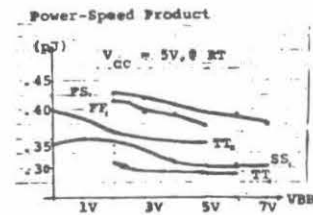


Fig. 1c - Performance of the ring-oscillator at 23 C, 5V VCC.

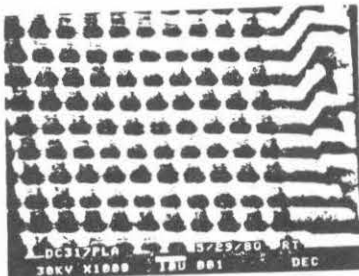


Fig. 2a SEM microphotograph of the NAND PLA array

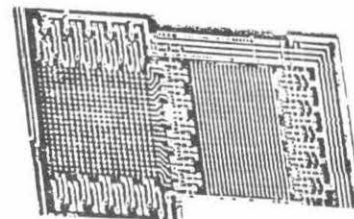


Fig. 2d - A 20x20 NAND PLA with 8μm/4μm devices.



Fig. 2b SEM microphotograph of the NAND PLA cells

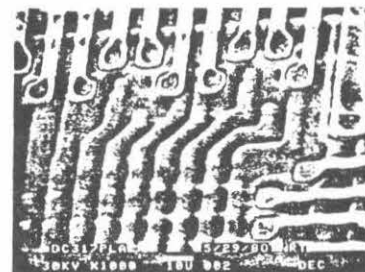


Fig. 2c - An example of the difficulty in interfacing to a small cell pitch of the NAND PLA

TABLE II

Basic Characteristics of NAND Circuit in Comparison with NOR Circuit [4]

Electrical Parameters	NOR	NAND
Logic Threshold *	$\frac{V_{DD}}{\sqrt{\frac{I_{pu}}{I_{pd}} \cdot \frac{W_{pu}}{W_{pd}}}}$	$\frac{V_{DD}}{\sqrt{\frac{I_{pu}}{I_{pd}} \cdot \frac{W_{pu}}{W_{pd}}}}$
Delay Time	$T_{NOR} \approx T_{DN}$	$T_{NAND} \approx n \cdot T_{DN}$
Pull-Down Device Width (to achieve same 'VOL')	$W_{NOR} = W_{DN}$	$W_{NAND} = n \cdot W_{DN}$

\* Assume same size for all pull-down devices.

\*\* Punch-thru is not a problem.

An Illustration for Minimum NAND Device Channel Width Needed for an Acceptable VOL

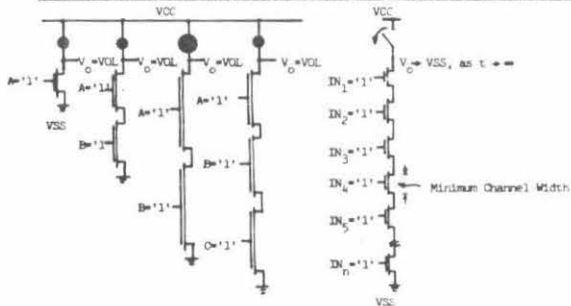


Fig. 3a

NAND Circuits with Static Pull-up

Fig. 3b

NAND Circuit with Dynamic Pull-up

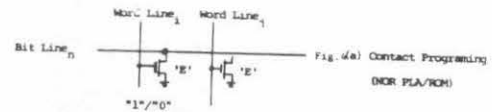


Fig. 4(a) Contact Programming (NOR PLA/ROM)

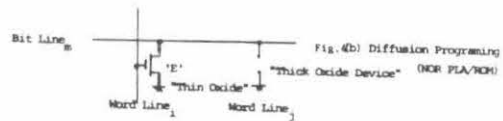


Fig. 4(b) Diffusion Programming (NOR PLA/ROM)

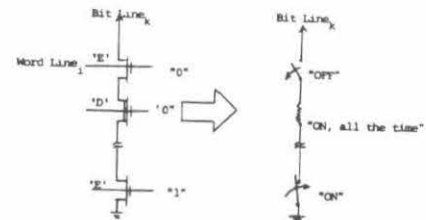
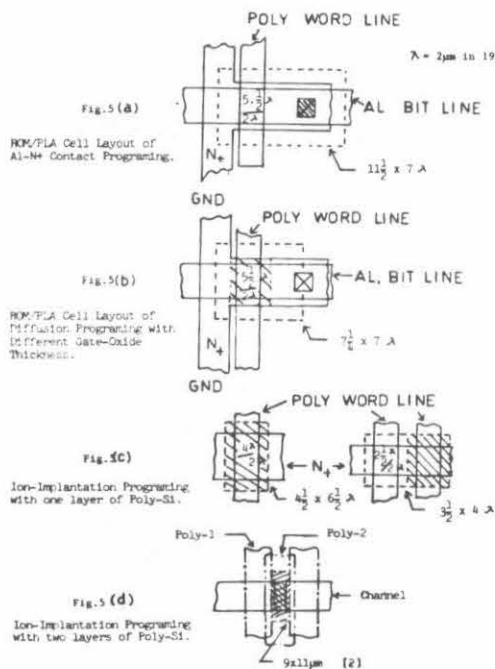


Fig. 4(c) Ion-implantation Programming Fig. 4(d) Switch equivalent diagram of NAND CRT



Reliability Study of the "Implant Programming" Cell:

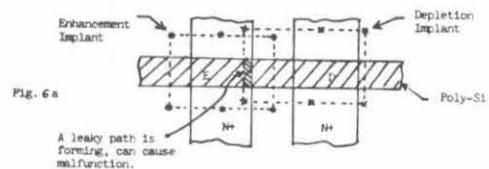


Fig. 6a

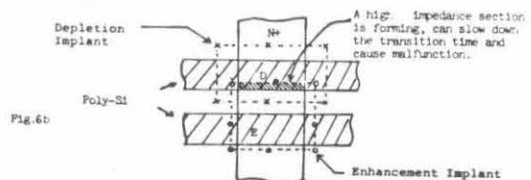


Fig. 6b

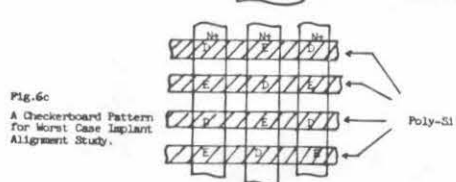


Fig. 6c

A Checkerboard Pattern for Worst Case Implant Alignment Study.



## PLA Design in NAND Structure

MODEL OF NAND CIRCUIT

As shown in Table II, the major reasons that the NAND circuit has not been used as popularly as the NOR structure are due to its inferior electrical properties in comparison with NOR circuit [4]. Although NAND circuit's d.c. characteristics, such as its logic threshold and pull-down device channel width, shown in Fig. 3(a), can be improved by using dynamic pull-up device, shown in Fig. 3(b). However, the delay time of a NAND circuit is slow and proportional to its number of transistors in series. Even so, a NAND circuit can use ion implantation as a programming method for its memory bits, as shown in Fig. 4, and this programming approach leads to the smallest PLA/ROM cell size that can be achieved by the current MOS technology.

CELL LAYOUT

For a conventional NOR Structure PLA/ROM Cell, as shown in Fig. 5(a), the size of a cell is defined and limited by its components--word line (poly), bit line (aluminum), contact between drain (N+) and bit line, memory transistor gate area, drain, and source (N+). Furthermore, due to process requirement, minimum area and space for each element must be used in the implementation of the memory cell. Thus, within the limitation of the present process technology, selection and/or elimination a certain part of the memory cell leads to different structure variations and cell sizes in PLA/ROM cell design. Table III shows the size ratio and then the basic properties of the four major types of cells which are well known to the public with straight forward layout techniques. From Fig. 5, it is obvious that the NAND structured PLA/ROM can be made up to a quarter of the size of a 'contact programming' cell, and the elimination of the contact also enhances the circuit reliability.

The size of the NAND Structure PLA/ROM cell can be made smaller if the process can provide smaller poly and N+ lines without causing electrical problems [3]. Furthermore, the alignment and resolution of the ion implant process hold the last barrier on the minimum size this approach can be.

When the NAND PLA/ROM cells are placed closer to each other, the enhancement and depletion implant can overlap into each other. Out of the four possible overlapping cases, there is one fatal case and the other three cases, although not fatal, can all cause electrical problems as shown in Fig. 6.

One way to test out the process and equipment limitations is using a checkerboard test pattern, shown in Fig. 6(c). This test pattern, if properly decoded, would be able to indicate the safety margins left in a process for implementing the NAND structure PLA/ROM. On the other hand, a NAND structured PLA/ROM is also a good test tool for process control monitoring, especially for ion implantation's definition control.

## PLA Design in NAND Structure

### CIRCUIT DESIGN CONSIDERATIONS

As shown in Table II, the NAND Structure PLA with static pull-up would have difficulties with ratio, pull-down device size, and slow speed in discharge against the constant conducting pull-up device. In dynamic operation, the ratio problem is eliminated, pull-down device size is minimized, and discharging time is reduced. However, the generation and implementation of the control clocks will complicate the design and require extra silicon area. As a result, in dynamic operation, the design effort and total implementation area for a PLA is more than putting two ROM arrays together. Furthermore, in NOR structure dynamic (or semi-dynamic) PLA design, an interface circuit is needed to allow precharging of the two arrays at the same time during the early cycle time of the operation. Consequently, a clock scheme of four phases generated from the system is the most common approach, and the safest way is using four non-overlapping clock phases to execute the operation at the expense of slow through-put time and zero process tracking capability.

For the proposed NAND PLA, the elimination of the interface circuit between the two arrays simplifies the layout work between the arrays and save area from implementing the interface circuit. Basically, a dynamic NAND circuit's access time is limited by its precharge and discharge time. In this proposed NAND PLA, precharge time is significantly reduced with precharge from both ends, because RC constant of the serial channel is halved. The precharging devices of the AND array are driven by bootstrapped voltage level, which gives the AND array full VCC level that helps to speed up the discharge time in the OR array. The precharge operation can be further optimized by generating a longer pulse width with normal VCC level for the OR array, because the OR array will be enabled only after the AND array is settled. A high beta ratio input inverter in the output register with amplified positive feedback through an intrinsic device, as shown in Fig. 8(g), allows a weak logic "1" output from the OR array at VCC-V<sub>Tn</sub> with no difficulty in the initial sensing and final stored level.

### CLOCKING SCHEME AND GENERATION

The proposed NAND PLA uses five clock phases. Their wave forms are show in Fig. 7(a), where C<sub>in</sub>, C<sub>pr</sub>, C<sub>ena</sub>, and C<sub>eno</sub> are used for the control of the PLA operation flow. While C<sub>la</sub> is designed for the latch of the processed data against precharge and low frequency operation's leakage problem. It depends upon each design's system spec and circuit structure, C<sub>la</sub> may be spared without effecting the performance.

TABLE III. ROM/PLA Cell Type and Structure

Type	No. Contact per Cell	Source Area shared by No. of Cell	Drain Area shared by No. of Cell	Programming Method	Cell Size Relative to Type a. cell	Comments
a	1	2	1	Contact	1.0	NOR Structure Fast Turn Around Time
b	1/2	2	1	Diffusion	2/3	NOR Structure Thin Gate Oxide- Selected Thick - Dis-selected
c	8	2	2	Ion Implant	1/3	NAND Structure Enhancement Implant- Selected Depletion Implant- Don't Care
d	8	2	2	Ion Implant	1/3	NAND Structure Two layers of poly-Si, yield is less than single Poly process.

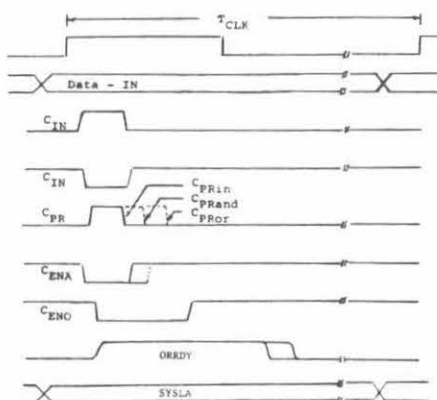


FIG. 7a CLOCK SCHEME AND WAVE FORMS FOR THE NAND PLA

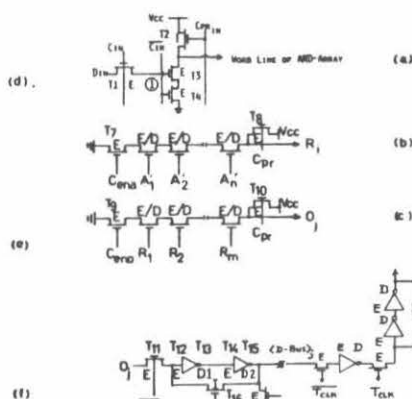
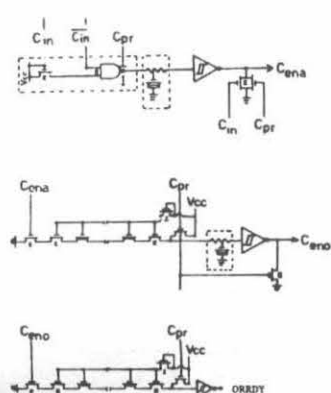


FIG. 8 - CIRCUITS USED IN THE NAND PLA

- a. Input-Buffer
- b. A cell of the AND-Array
- c. A cell of the OR-Array
- d. Cena generator and dummy input-Buffer
- e. Ceno generator
- f. Cisa generator
- g. Output-Register

## PLA Design in NAND Structure

All these clocks are derived from the rising edge of the input system clock, Tclk, and they are generated through dummy circuits which provide enough tracking capability against process variations and power supply changes. In order to use the same circuits or cells to achieve the best result for dummy circuits, the proper number of depletion devices are suggested to use for capacitance loading duplication, and device width effect {3} is another source to provide extra operation margin.

### CIRCUIT OPERATION:

#### 1. Array precharge--

Cpr is a self-timed clock pulse triggered by the rising edge of the system clock, Tclk. As shown in Fig. 9, both Cpr and Cin are generated through a dummy circuit which uses a row of the OR array to track the loading in OR array and uses a column of the AND array to track the completion of the precharge action. During this period, both Cena and Ceno are '0'. This allows both arrays to be fully charged to their highest possible level against couplings and the so-called "charge sharing" problem (3).

#### 2. Data input--

New input data should be ready in the beginning of a new cycle. Those input data are strobed into the input buffer through T1, see Fig. 8(a), and temporarily stored at node ① after the Cin pulse is gone. During the precharge time, T4 is turned 'off' by Cin. This guarantees the completion of the precharge to be independent of those input data--such that either T3 is 'on' or 'off', there is no d.c. path through T2 to ground due to the exclusive wave forms between Cpr and Cin. The input buffer only consists of 4 transistors. Transistor T2 is an intrinsic device which gives a higher output voltage than an enhancement device when Cpr is high, but it will not conduct much current, with proper choice of channel length, when Cpr is low. This structure allows optimal output level and minimum device sizes for the pull-down devices T3 and T4, as well as for T2 itself, partly because the intrinsic device has the highest mobility among the four types of devices available in this process. Also, due to the compactness of the structure and the lack of appreciable d.c. path in this input buffer, the interface problem is helped and power dissipation is greatly reduced.

#### 3. Enabling of the arrays--

Once the arrays are fully charged, Cpr and then Cin go down to '0'. When Cin is '1', T4 in the dummy input buffer, INdmbf, turns 'on', and the output of the INdmbf, see Fig. 8(d), start to change from a precharged '1' to '0' due to the input is VCC. With enough depletion type capacitors on the output and a Schmitt trigger to sense the change, Cena is ensured to turn 'on' only after the completion of all the inverted input data have been transferred to their outputs, or word lines of the AND array.

## PLA Design in NAND Structure

Cena enables the AND array to decode its inputs through its pre-programmed memory bits. Cena also enables a dummy circuit in the AND array to discharge from its precharged level to '0' and thus generate the Ceno pulse, as shown in Fig. 8(e), by the same principle as the Cena generation. With the starting of the Ceno clock pulse, NAND PLA is ready to send out its decoded results to the output register and outside buses.

## 4. Output Register--

As shown in Fig. 8(g), the output buffer contains 7 transistors with static pull-ups used in the register to provide easy data storage through amplified positive feedback. The advantage of precharging the output data bus lines is incorporated into the circuit design to save power and size in the output section. Because of the precharge from T17, T15 can be made small as D1 (light depletion) device for sustaining purpose. This also speeds up the discharge on the bus line if  $\overline{\text{Dout}}$  is a '0'.

Even though the output section only contains a minimum number of transistors, the layout work to interface the OR array cell pitch to the output buffer is not an easy task. Techniques like: combining two buffers together; bringing out output from both ends of the OR array; or constructing the buffers at a distance and then connect the two parts through spread out N+, Poly, or metal lines, are up to the designer's choice for the best matching between the NAND PLA and system requirement.

## 5. Latch of the output register--

For a dynamic PLA without any sustaining pull-up device used in the arrays, maintaining  $\text{Cena} = \text{Ceno} = '1'$  to keep ORRDY at '0' is needed against noise and coupling. Furthermore, isolating the OR array precharged outputs from their output buffers is also crucial for operation at low frequency where leakage eventually will change a precharged '1' to a '0'. In this proposed NAND PLA, a simple but effective design is used, as shown in Fig. 7(b). The dummy circuit in the OR array generates a delayed "data is ready" signal, ORRDY, which tracks after the completion of the OR array transition through narrower device width transistors and/or a Schmitt trigger. The Cla clock is also controlled by a system latch signal, SYSLA, which happens only after the transition is over, see Fig. 7(b).

## PLA Design in NAND Structure

TEST RESULTS

Since most of the circuits proposed for the NAND PLA in this paper are of dynamic operation, power dissipation is being optimized to a minimum. Thus, the major concerns left for this approach are speed related device width effect, gate height effect, circuit operating range, and effect of precharge methods [3]. These measured data are considered to be useful as a reference point in related applications with this kind of circuit structure.

FURTHER IMPROVEMENT

**SPEED:** If the negative '0' level can be generated from an external VBB power supply, a Depletion-1/Depletion-2 pair can be used for a transistor programming purpose. A D1 device, even with the lowest carrier mobility among the four types of devices, does conduct current more strongly at the same gate voltage than the enhancement type, thus the D1/D2 pair would be faster in transition time than the regular E/D2 pair.

**DENSITY:** Since this structure does not need metal lines in the array, this PLA's memory area is free for metal lines of other functions on the chip, as shown in Fig. 10. If this PLA section is properly located on the chip, further area savings can be achieved through sharing the memory area with wide power lines or limited number of data/control lines. The problem of poly lines going across the N+ lines can be handled by using depletion implant at the cross section. Properly clocked poly line, buried contact, and a few more contact points to the power lines will further improve the topology and electric conditions in this special application.

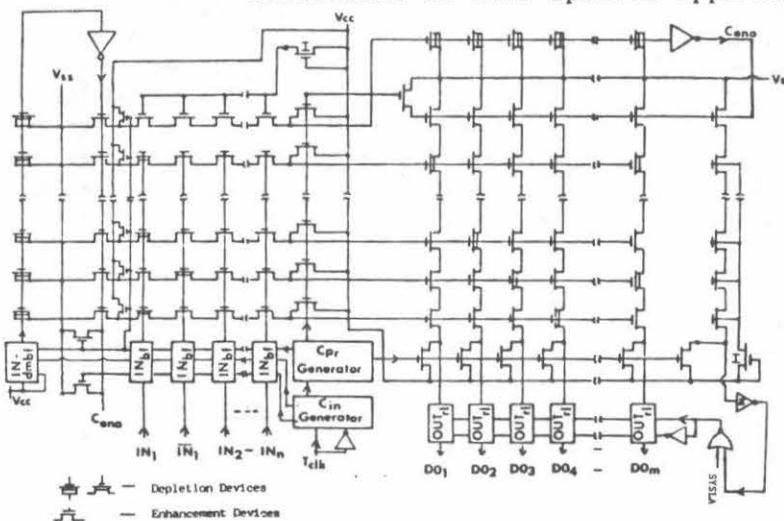


Fig.7b - The Circuit Structure of the NAND PLA.

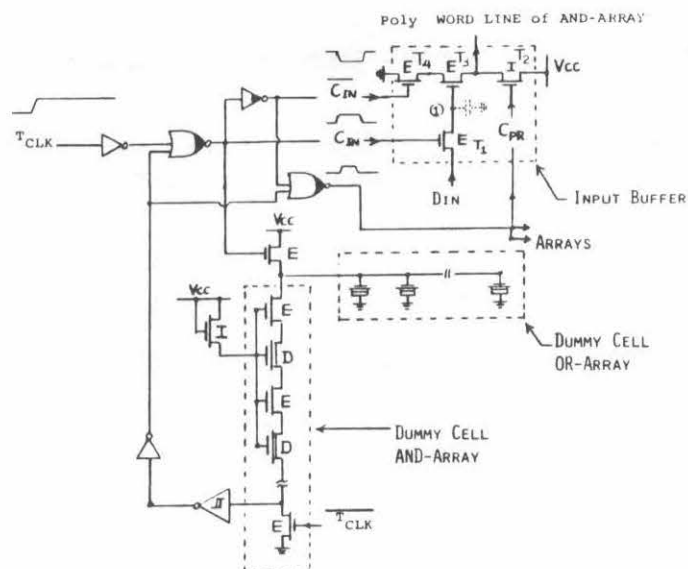


FIG. 9 Input-Buffer and the generation of the self-timed clocks  $C_{IN}$  and  $C_{PR}$ .

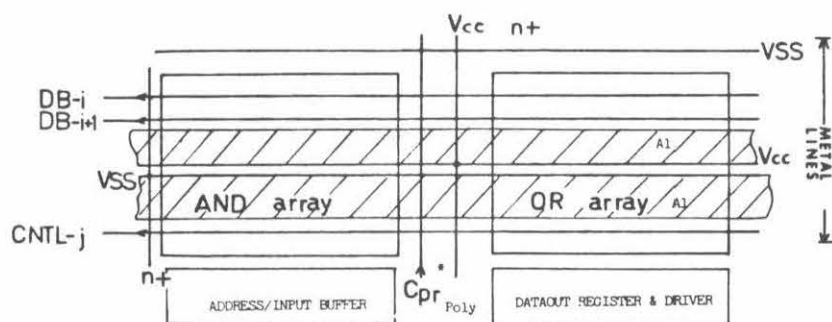
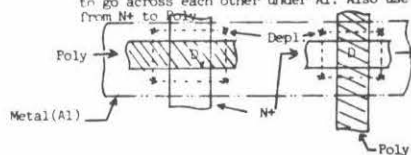


Fig. 10 PLA Memory Area is FREE for Metal Lines!!!

- Use Depletion Ion-implant to allow Poly and N+ to go across each other under Al. Also use B.C. from N+ to Poly.





## ACKNOWLEDGEMENTS

The author would like to thank those whose contribution and help made this work possible.

L. Nguyen	--His request for development of an advanced VLSI Chip.
D. Morgan	
Tuan H.T. (BURROUGHS)	--Their technical evaluation and encouragement.
J. Zeh	--His vision and commitment on the project.
J. Schneider	--His decision, funding and continuous support.
T. Northrup, K. Slater, and F. Zereski	--Their continuous encouragement and support.

---

Layout	--H. Nguyen, J. McHood, M. Riley and A. Sella
Process	--L. Y. Wu
P400 Program	--R. Spencer
Test	--R. Saul and D. Cote

Review and preparation of the manuscripts, figures and typing--M. Forsyth, J. Blake, A. Flohr, J. Giles, R. Ryan, and M. Burton.

---

## REFERENCES

1. H. Kawagoe and Nobuhior Tsuji, "Minimum size ROM structure compatible with silicon-gate E/D/MOS LSI," IEEE J. Solid-State Circuits, Vol. sc-11, No. 3, pp. 360-364, June 1976.
2. Y. Kitano, S. Kohda, H. Kiduchi and S. Sakai, "A 4Mb full wafer ROM," in ISSCC Dig. Tech. Papers, Feb. 1980, pp. 150-151.
3. Chong M. Lin, "A 4um NMOS NAND Structure PLA," IEEE J. Solid-State Circuits, April 1981.
4. Carver Mead, Lynn Conway and Charles L. Seitz, Introduction to VLSI systems, Addison-Wesley Publishing Co., 1980, pp. 15-16, and Ch. 7.



A MULTIPROJECT CHIP APPROACH TO THE TEACHING OF  
ANALOG MOS LSI AND VLSI

Yannis P. Tsividis\* and Dimitri A. Antoniadis

Department of Electrical Engineering and Computer Science  
Massachusetts Institute of Technology  
Cambridge, MA 02139

Abstract

Multiproject chip implementation has been used in teaching analog MOS circuit design. After having worked with computer simulation and layout aids in homework problems, students designed novel circuits including several high performance op amps, an A/D converter, a switched capacitor filter, a 1 K dynamic RAM, and a variety of less conventional MOS circuits such as a V/I converter, an AC/DC converter, an AM radio receiver, a digitally-controlled analog signal processor, and on-chip circuitry for measuring transistor capacitances. These circuits were laid out as part of an NMOS multiproject chip. Several of the designs exhibit a considerable degree of innovation; fabrication pending, computer simulation shows that some may be pushing the state of the art. Several designs are of interest to digital designers; in fact, the course has provided knowledge and technique needed for detailed digital circuit design at the gate level.

1. INTRODUCTION

During the last few years the development of MOS IC design has advanced in two fronts. On one hand improvements in fabrication have made possible the implementation of LSI and VLSI digital systems. On the other hand, introduction of analog MOS circuit techniques has made possible single chip integration of high performance analog and analog/digital circuits (1) such as A/D and D/A converters, PCM encoders and decoders and a variety of other telecommunication systems, switched capacitor filters, microcomputers with analog interfaces, several special purpose signal processors, and high performance operational amplifiers.

---

\*On leave from the Department of Electrical Engineering, Columbia University, during the Fall of 1980.

Courses and research projects in digital LSI and VLSI have been initiated in many universities. A particularly successful teaching and research vehicle is the incorporation of a number of diverse design projects on a single chip. This economical and speedy realization of new circuits and the associated design methodology have come to be called the "multiproject chip" approach (2,3). This approach has proven its value in the classroom by allowing students to be exposed to all aspects of integrated circuit design, layout and experimental evaluation of their projects. This paper describes the use of the multiproject chip approach in the teaching of a one-semester course on analog MOS circuit design, offered at MIT during the fall of 1980, with very encouraging results. The course has evolved from a similar course, taught over the last few years at Columbia University, which included the design project but not the multiproject chip implementation.

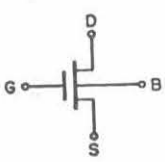
## 2. COURSE OUTLINE

The course to be described is at the senior/graduate level. The assumed background is a one year junior electronics sequence (in the first offering at MIT it happened that most students in the class were graduate). No background in MOS devices and circuits is assumed. A list of the major topics covered follows:

- Semiempirical MOS transistor model
- Fabrication and computer aided layout
- Basic circuit building blocks
- Computer aided circuit analysis
- Operational amplifiers
- Large signal consideration (transient response and distortion)
- Noise
- Voltage reference sources
- Comparators
- A/D and D/A converters, PCM encoders and decoders
- Switched capacitor filters
- Detailed device physics and higher order models

The topics have been selected so as to give working knowledge for the design of high performance analog MOS circuits; in fact the design and layout of such circuits is a required project in the course. The particular order in which the topics are presented is chosen in order to allow an early start of the design project which has to meet specific deadlines associated with the multiproject chip implementation. This makes it necessary to begin the course with the presentation of a semiempirical device model which has to be taken temporarily for granted; enough plausibility arguments are presented so that the model "makes sense", and students are promised a detailed derivation from physical principles in the last part of the course. Judging from student responses to a questionnaire, this does not cause problems. Both NMOS and CMOS circuits are covered.

The standard square law equations, used for strong inversion in hand analysis of digital circuits, are often inadequate for analog design especially when the substrate doping is relatively high. A more accurate, yet simple, set of equations (4) used in the course appears in Fig. 1 (this model is modified for short channel devices to include channel length modulation).



$$I_D = K \left[ 2(V_{GS} - V_T)V_{DS} - (1+\delta)V_{DS}^2 \right], \quad V_{DS} \geq \frac{V_{GS} - V_T}{1+\delta}$$

$$= \frac{K}{1+\delta} (V_{GS} - V_T)^2, \quad V_{DS} \leq \frac{V_{GS} - V_T}{1+\delta}$$

$$K = K' \left( \frac{W}{L} \right)$$

$$V_T = V_{T0} + \gamma (\sqrt{V_{SB} + \phi_B} - \sqrt{\phi_B})$$

FIGURE 1 Semiempirical DC model for long channel devices.

$K'$  is a process dependent parameter and  $\delta$  is a parameter which depends on substrate doping and substrate bias. However, the quantity  $1+\delta$  is only weakly dependent on substrate bias and to first order can be considered constant for a given device. Both  $K'$  and  $1+\delta$  are empirically determined by fitting experimental I-V curves. The parameters  $V_{T0}$ ,  $\gamma$  and  $\phi_B$  in the threshold voltage expression are also empirically determined. A small signal equivalent circuit is derived from the above set of DC equations; when intrinsic device capacitances and the drain small signal conductance are added to it, the circuit of Fig. 2 is obtained (5,6,7). Junction and overlap capacitances are easily added to this model. The above models represent good compromises between accuracy and simplicity for hand analysis. Students are made aware of higher order models, but a detailed discussion of these is postponed until the last part of the course.

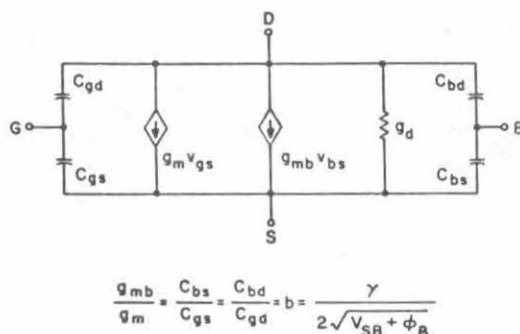


FIGURE 2 Small-signal model.

Both computer aided layout and circuit analysis are taught in conjunction with homework problems at which students are exposed early to the use of computer aids; in our case the programs AIDS (8) and SPICE (9) were used on a DEC-20/60 computer.

Basic circuit building blocks are presented at a great level of detail. Bias, low frequency small signal operation and high frequency considerations

are discussed for inverters (CMOS, depletion load NMOS and enhancement mode NMOS), source followers, differential stages, cascode stages, current sources and level shifters. The knowledge thus gained is used to discuss more complex circuit blocks. The students are at this point ready for a detailed exposure to various operational amplifier configurations; bias and small signal calculations for working operational amplifiers are treated at length. High frequency considerations and frequency compensation are emphasized. At about this point students start their project work; details are given in Section 3.

The lectures continue with the topics of transient response, distortion and noise. The approach used for noise analysis is that of Ref. (10). Voltage reference sources, comparators, A/D and D/A converters, and PCM codecs are then discussed and examples of working designs from the literature are analyzed. The emphasis in the treatment of switched capacitor circuits is on basic principles. Exact analysis is taught using the intuitively appealing concept of charge conservation within closed surfaces not crossed by conductors (11). No matrix analysis is used. The students are cautioned against carelessly using resistive equivalents of switched capacitors and illustrative examples of misuse of such equivalents are presented. Although several working filter designs are discussed, not much time is devoted to the synthesis of such circuits as it is felt that this topic is better left to a course on network synthesis.

The final topic discussed in the course is that of MOS transistor physics and models. Potentials within the semiconductor instead of energy bands are used in such a way that rigor is not compromised. Both the semiempirical model discussed above, and more accurate models are derived from first principles. Small geometry and high order effects are discussed and general capacitance and charge models are introduced.

In the MIT offering weekly homework assignments were given during the first part of the course; later these were gradually phased out to allow more time for work on the design project. A total of 8 assignments were given during the semester. Student performance was judged from the design project,

homework and personal interaction with the instructor. No midterm or final examination were given. Previous offerings of this course at Columbia University have included such examinations.

### 3. THE DESIGN PROJECT

Independently of whether it is finally realized on silicon, the design project provides the student with an opportunity to pull together what he has learned on circuit design. Realization of the circuit as part of the multiproject chip offers the additional opportunity to go through the remaining steps typical in an industrial environment, these being layout and final evaluation in the lab; it also serves as an important booster to student motivation. Because it is impossible to have the chips fabricated before the end of the semester, the evaluation in the lab cannot be a required part of the course; however, experience with other courses involving a multiproject chip has shown that the majority of the students return the following semester, on their own initiative, to evaluate their circuits.

The implementation of the MIT multiproject chip is managed by MOSIS at the University of Southern California; the chip is to be fabricated by the Integrated Circuits Laboratory, Hewlett Packard, Inc., Palo Alto. A NMOS enhancement-depletion single-level polysilicon process is to be used, with nominal substrate doping of  $6 \times 10^{14} \text{ cm}^{-3}$ . The nominal threshold voltages at 0 substrate bias are +1 V and -4 V for the enhancement and depletion transistors, respectively. The layout rules followed are those in reference (2), with  $\lambda = 2.5 \text{ } \mu\text{m}$ . Minimum channel dimension for the projects was set at  $3\lambda$ , as opposed to  $2\lambda$  used in digital projects, to avoid modeling inadequacies at short and narrow channels. This was necessary because of lack of appropriate test transistors for detailed characterization. Polysilicon-to-depletion implant capacitors are used as the above process does not permit the implementation of higher quality structures. More appropriate processes for the purposes of this course are double-level polysilicon NMOS or CMOS, or at least a modified single-level polysilicon process that would allow the fabrication of reasonable value high quality capacitors between metal and polysilicon; although of lower performance, metal-gate processes can also be used.

After the first third of the semester students were asked to submit a brief proposal outlining the design project they intended to work on. A high performance operational amplifier was suggested as a possible project by the instructor, and a set of state-of-the-art specifications that had to be met or exceeded was given. Minimization of power consumption was emphasized as one of the most important design goals. Recently designed high-performance operational amplifiers in the industry were claimed to have a power dissipation of only 0.75 mW, so this was set as a specification to be bettered. The students were asked to work in groups of two or more in order to reduce their work load, facilitate supervision and avoid overloading the computer facilities. All student designs were simulated using the program SPICE. The students were supplied with model parameters, which were derived from the information we had on the process to be used. Unfortunately, no appropriate test devices were available for detailed characterization, so we had to use instead devices integrated using a related process and then extrapolate the results. A further complication arose from the fact that we had no previous experience with the model in the particular version of the program SPICE we used. However, every effort was made to use as reasonable a set of model parameters as possible, and it is hoped that simulation results are a good indication of what will be seen in the laboratory when the fabricated chips are received. The design projects undertaken by the various groups are listed below:

- Low power enhancement/depletion operational amplifier (5 groups)
- Low power enhancement-only operational amplifier (3 groups)
- High speed operational amplifier
- Autozeroing operational amplifier
- A/D converter
- AC/DC converter
- Voltage to current converter
- Switched capacitor filter
- Digitally programmable analog filter
- 1 Kbit dynamic RAM
- On-chip circuitry for MOS transistor capacitance measurement
- AM radio receiver



Some of the student designs will be briefly described below, and representative computer simulation results will be quoted. As can be seen in the list given above, the most popular design project was that of a low-power enhancement-depletion operational amplifier. Five groups have designed such circuits for operation from  $\pm 5$  V power supplies, with power dissipation ranging from 0.4 mW to 1 mW. Low frequency gains are between 60 dB and 74 dB, and unity gain frequencies after frequency compensation is between 0.65 MHz and 3 MHz. The 1% settling times are between 0.7  $\mu$ s and 4  $\mu$ s for a 10 pF load. An example of a student design is shown in Fig. 3. Another design uses an architecture drastically different from that of any NMOS operational amplifier presented to date; the students who designed the circuit have asked us not to present it because they plan to apply for a patent.

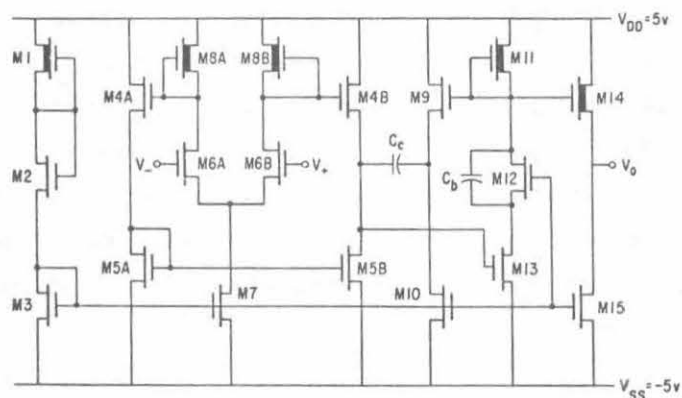


FIGURE 3 A low-power enhancement/depletion operational amplifier (M. Elbuluk and J. Harrison).

One group has decided to meet the challenge of using only enhancement devices in their operational amplifier; they have come up with the circuit of Fig. 4, and a performance certainly impressive for an all-enhancement design: a power dissipation of 0.93 mW, a low-frequency gain of 61 dB, a unity gain frequency of 420 KHz, and a 1% settling time of 2.3  $\mu$ s with a 10 pF load. The circuit is more process-insensitive and has much lower distortion than most enhancement-depletion operational amplifiers.



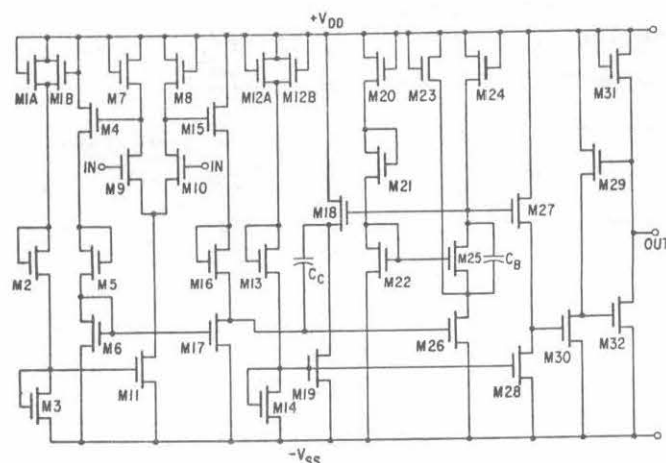


FIGURE 4 A low power enhancement-only operational amplifier (C. C. Cederberg and B. V. Karlsson).

Three groups have designed high speed operational amplifiers. The simplest and fastest design is shown in Fig. 5; it compromises gain, which is only 40 dB, for speed. The unity gain frequency is 126 MHz, and the 1% settling time is only 24 ns with a 5 pf load charged through a series device. Power dissipation is 15 mW.

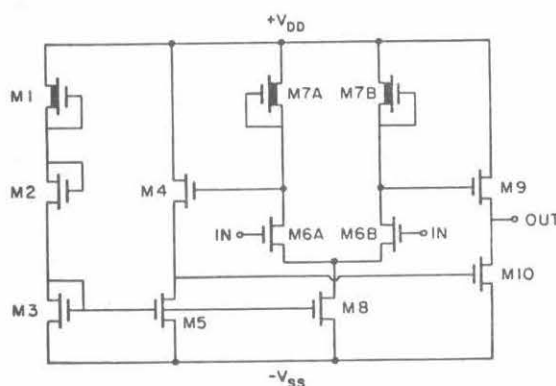


FIGURE 5 A high speed, low gain operational amplifier (C. Christensen and W. Shiley).

The autozeroing operational amplifier's concept is shown in Fig. 6; the switches are implemented with MOS transistors. The bottom amplifier is used to null alternatively the offset of itself and that of the top amplifier. This avoids the problems of commutated auto-zero designs, where at every commutation the signal output must slew from the offset value to the signal value.

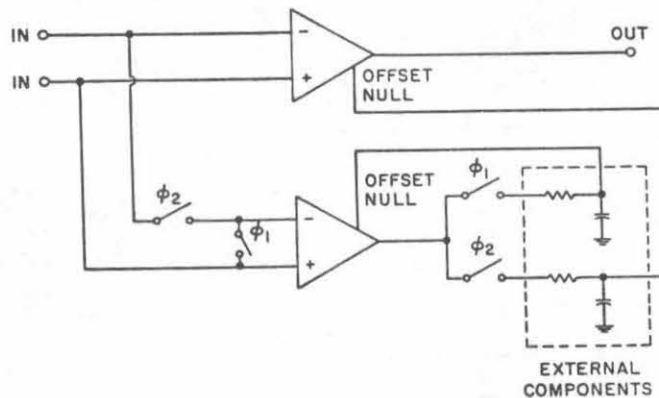


FIGURE 6 An autozeroing operational amplifier (M. Coln).

The A/D converter project employs charge redistribution using three capacitors; the analog part of the design is shown in Fig. 7. It is expected that it will perform an 8-bit conversion in 27  $\mu$ s.

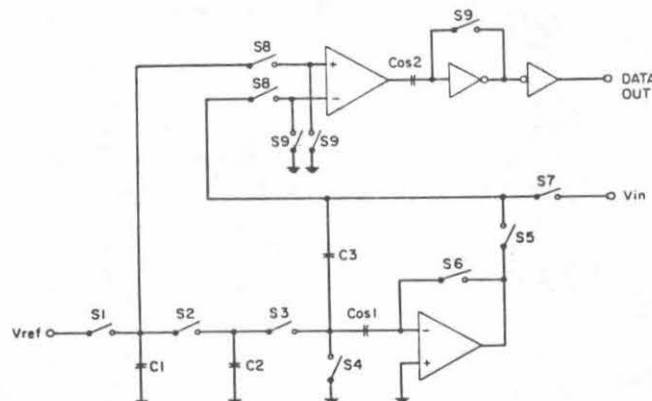


FIGURE 7 An 8-bit, successive approximation A/D converter (S. McCormick and A. Garcia).

Another project aimed at instrumentation applications is the voltage-to-current converter of Fig. 9. One of the uses of this circuit is in developing a temperature-insensitive, supply-insensitive reference current from an existing reference voltage. The output current is produced from an internal reference current through a mirror circuit. Internal feedback circuitry adjusts the reference current until it charges a capacitor to a voltage equal to the voltage applied externally, in a specific amount of time determined by an external clock.

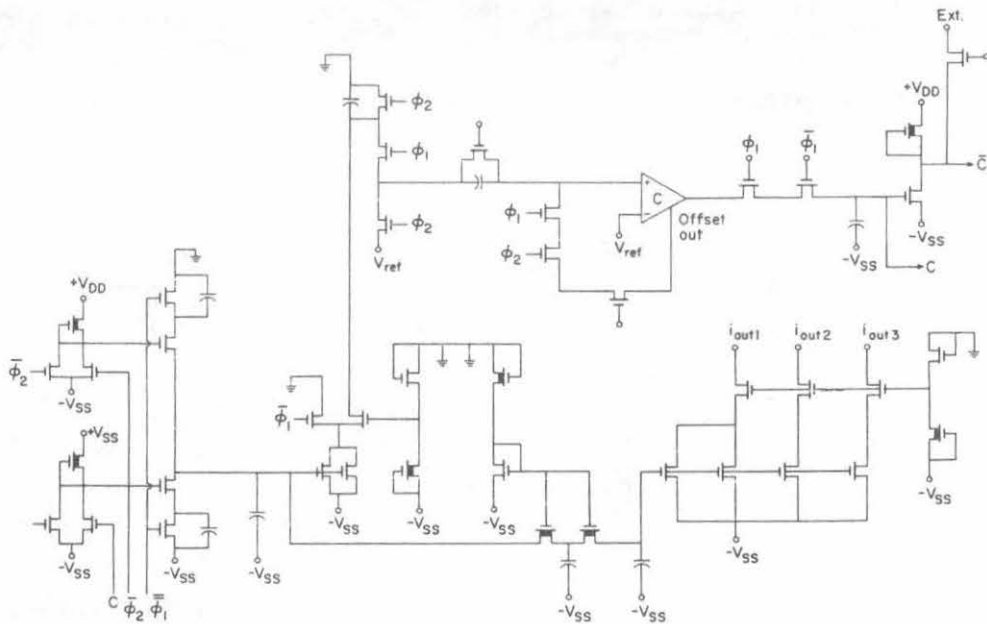


FIGURE 9 A voltage-to-current converter (M. M. Colavita and F. L. Terry, Jr.).

One project dealt with the design of the switched capacitor biquadratic notch filter shown in Fig. 10; the topology is that reported in (12). This design has been adjusted for minimum capacitance spread and minimum total capacitance. A very good operational amplifier was part of the design; in fact many of the "non-op amp" projects actually contained good operational amplifiers.

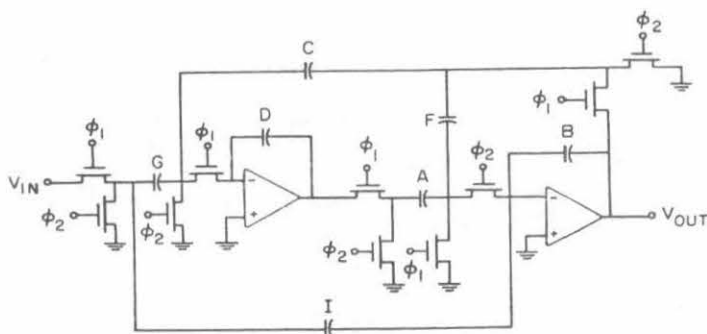


FIGURE 10 A switched capacitor biquad notch filter (M. M. DeSmith and D. W. Duehren).

The digitally programmable analog filter project undertaken by C. W. Mangelsdorf and A. L. Robinson uses pulse width control to adjust the transfer function coefficients; the value of these coefficients depends only on timing and is independent of element values or even element value ratios for some configurations (13). The circuit is laid out in such a way that both transversal and recursive filters can be implemented. The design includes many sample-and-hold circuits, which share two operational amplifiers using the technique described in (14).

A good example of the extensive analog design involved in realizing digital circuits at the gate level is a dynamic RAM; one of the projects was the design of a one-transistor-cell, 1 Kbit dynamic RAM including sense amplifiers, word line driver and column decoder. Its density is comparable to that of INTEL 2104. According to the students, one of the most challenging parts was the high voltage driver, shown in Fig. 11.



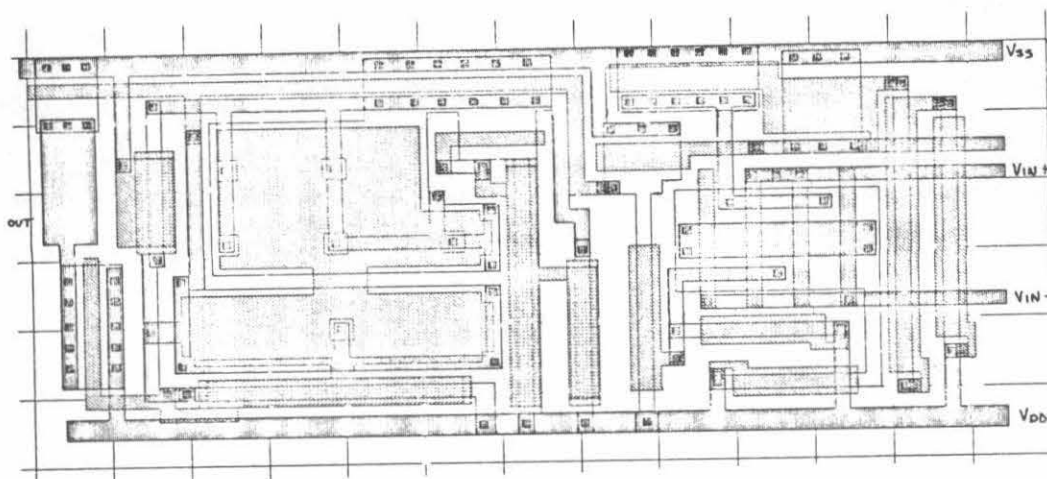


FIGURE 12 Layout of an operational amplifier (D. Goddeau and M. Johnson).

Student time spent on project work was a large part of the total time spent on the course; answers to a questionnaire distributed in class show that the average time spent for the course per week was 15 hours. We are looking for ways to decrease this time in future offerings, without subtracting significantly from the value of the course.

#### 4. CONCLUSIONS

A one-semester course on analog MOS design has been taught several times at Columbia University and recently at MIT. The course covers topics from device physics and models to detailed circuit and layout, and one of its important parts is a state-of-the-art design project in which students put together what they have learned in the lectures. In the last offering of the course (MIT), the emphasis on the project was increased; student designs will be implemented as part of a multiproject chip. Computer simulation results show that many designs are of excellent quality and are innovative; some may be pushing the state-of-the-art. Several designs are of interest to digital designers; in fact, the course has provided knowledge and technique needed for detailed digital circuit design at the gate level.

## 5. ACKNOWLEDGEMENTS

The authors would like to thank the many people at MIT who have contributed to the success of the course; among them Paul Penfield, Jr. for useful discussions and for giving the lecture on layout aids, Steve Senturia for useful discussions, and Zahid Ansari, Mark Johnson, John Paulos and Irfan Rahim for characterization of devices. Thanks also go to the many students, both at Columbia and MIT, who have helped shape the course with their helpful comments and questions.

## REFERENCES

1. See special issues on analog circuits, IEEE Journal of Solid-State Circuits, December of each year starting with 1975.
2. C. Mead and L. Conway, "Introduction to VLSI Systems," Addison-Wesley, 1980.
3. L. Conway, "University Scene," Lambda Magazine, vol. 1, no. 3, pp. 65-69, fourth quarter 1980.
4. G. Merckel, J. Borel and N. Z. Copcey, "An Accurate Large-Signal MOS Transistor Model for Use in Computer-Aided Design," IEEE Trans. on Electron Devices, ED-19, p. 681, 1972.
5. Y. P. Tsividis, "Design considerations in single-channel MOS analog integrated circuits - A tutorial," IEEE J. Solid-State Circuits, vol SC-13, pp. 383-391, June 1978.
6. Y. P. Tsividis, "Relation between incremental intrinsic capacitances and transconductances in MOS transistors," IEEE Transactions on Electron Devices, vol. ED-27, pp. 946-948, May 1980.
7. J. E. Meyer, "MOS models and circuit simulation," RCA Review, vol. 32, pp. 42-63, 1971.



8. P. Penfield, Jr., "AIDS-79 User's Manual," Integrated Circuit Memo No. 80-14, Department of Electrical Engineering and Computer Science, MIT.
9. L. W. Nagel, "SPICE 2: A computer program to simulate semiconductor circuits," ERL Report No. ERL-M520, University of California, Berkeley, 1975.
10. J. C. Bertails, "Low-frequency noise considerations for MOS amplifier design," IEEE J. Solid-State Circuits, vol. SC-14, pp. 773-776, August 1979.
11. Y. P. Tsividis, "Analysis of switched capacitive networks," IEEE Transactions on Circuits and Systems, vol. CAS-26, pp. 935-947, November 1979.
12. P. E. Fleisher and K. R. Laker, "A family of active switched capacitor biquad building blocks," Bell System Technical Journal, vol. 58, pp. 2235-2269, December 1979.
13. Y. P. Tsividis, "Method for signal processing with transfer function coefficients dependent only on timing," Electronics Letters, vol. 16, pp. 796-798, 9th October 1980.
14. L. Bienstman and H. J. DeMan, "An eight-channel 8-bit microprocessor compatible NMOS converter with programmable scaling," IEEE J. Solid-State Circuits, vol. SC-15, pp. 1051-1059, December 1980.



*DESIGN DISCIPLINES SESSION*

*Chairperson: MARTIN REM  
Professor of Mathematics  
Technical University  
Eindhoven, The Netherlands  
Visiting Professor of Computer Science  
Caltech*

## DESIGN DISCIPLINES SESSION

The management of complexity is among the most important problems associated with the design of computing structures, and with the design of VLSI structures in particular.

The aim of establishing good design disciplines is the raising of our confidence in the products we design. Such disciplines will of necessity involve an abstraction from the physical properties of the underlying VLSI medium. They involve the construction of mathematical models in which we design our computations and that allow us to prove properties of our computations. The intellectual mastering of VLSI design requires the cooperation of mathematical methods, circuit design, and programming methodology. In this session we get an impression of what it is that is making this meeting ground so exciting.

There are two aspects any mathematical abstraction in VLSI design must never neglect:

(1) The notation in which we express our designs, our "programming language," should have a rigorous definition. Only then will we be able to consider correctness proofs and formal verification. In order not to be overwhelmed by the inherent complexity of the VLSI medium the notation should support (or: force) the hierarchical structuring of designs. The first two papers address this topic. The first gives a mathematical treatment of VLSI arrays. The second paper discusses a hierarchical notation for computations that are to be realized as CMOS circuits.

(2) The mathematical model, the universe in which we express our designs and that we use to argue about our designs, should be based on underlying physical properties. The other four papers emphasize this aspect. The third paper of the session addresses the mappability of topological structures into the planar VLSI medium. The next two discuss the topic of signal propagation delays. In the session's fourth paper it is shown that under certain conditions the delay time may be assumed to be a logarithmic function of the wire length. When these conditions are not met one has to resort to linear delay times. A complexity model under the linear time assumption is the subject of the fifth paper. The last paper of the session discusses a variant of switching theory that is more appropriate to VLSI than the traditional theory.

# Towards a Formal Treatment of VLSI Arrays

Lennart Johnsson  
Computer Science  
California Institute of Technology  
Pasadena, CA 91125  
and  
Information Sciences Institute

Danny Cohen  
Information Sciences Institute  
Marina Del Rey, CA 90291  
and  
California Institute of Technology

Uri Weiser  
Computer Science  
University of Utah  
Salt Lake City, UT 84112

Alan L. Davis  
Computer Science  
University of Utah  
Salt Lake City, UT 84112

## Abstract

This paper presents a formalism for describing the behavior of computational networks at the algorithmic level. It establishes a direct correspondence between the mathematical expressions defining a function and the computational networks which compute that function. By formally manipulating the symbolic expressions that define a function, it is possible to obtain different networks that compute the function. From this mathematical description of a network, one can directly determine certain important characteristics of computational networks, such as computational rate, performance and communication requirements. The use of this formalism for design and verification is demonstrated on computational networks for Finite Impulse Response (FIR) filters, matrix operations, and the Discrete Fourier Transform (DFT).

The progression of computations can often be modeled by wave fronts in an illuminating way. The formalism supports this model. A computational network can be viewed in an abstract form that can be represented as a graph. The duality between the graph representation and the mathematical expressions is briefly introduced.

## Introduction

This paper addresses the problem of formally describing the behavior of computational networks at the algorithmic level. The focus is on the correspondence between equations defining a certain computation and networks performing that computation. In an equation there may be no concept of time. An equation expresses how a variable is related to a set of constants, other variables, and possibly itself. Time is, however, intrinsically associated with any computation performed by a physical device (Mead and Conway [10]).

VLSI technology promises to offer substantial computational power. With submicron technology, on the order of a million to ten million transistors can be placed on a single chip. The complexity of designing such a chip is orders of magnitude greater than that typical today. The need for proper abstractions at all levels of design is apparent. These abstractions have to be consistent so that a higher level description can be expanded in a hierarchical manner to levels where a direct mapping to silicon will generate circuitry that performs the function of a high-level description. One hierarchical approach, Rowson [12], not only brings the complexity of VLSI circuit design within reach of humans but also enhances correctness. The approach is also a good basis for advanced design tools such as silicon compilers (Ayres [1] and Johannsen [5]). Several silicon compilers exist today. The first silicon compiler, based on a special class of floorplans, has been followed by less restricted compilers. Currently available compilers accept an input at the register-transfer level.

Chen and Mead [2] have proposed a notation for designing concurrent systems. Their notation supports a hierarchical approach towards system design and enhances proof of liveness and safeness of concurrent systems. In mapping a behavioral description to silicon, it is also necessary to insure that the constructs used have a correspondence in circuits with an electrical behavior matching the description. Rem and Mead [11] have proposed a notation and composition rules that insure a correct correspondence between a syntactically correct behavioral description and the behavior of circuitry that can be generated from the description.

Early in the design of a computational network, a decision must be made about how the data required by the computation will enter the network. In a real-time signal-processing environment, a variable is typically observed (sampled) at discrete times, often at a constant frequency. In such applications it is natural to assume that the order of the data input to the network is the same as the order of the generation of the data. If the input data resides in a random access memory, then there are several ways for the data to be organized without requiring additional hardware or adversely affecting performance. The organization of the input data is, however, of prime importance for the size,

structure, and performance of the computational networks that can be designed to compute the desired function.

For simplicity, the derivations of the networks presented here are assumed to be synchronous. The results derived apply also to self-timed design (Seitz [13]). In the computation of most functions, not all computations can be performed concurrently. Sequence requirements implied by the algorithms used to compute a function have to be satisfied in order to obtain a correct result. It will be seen that the computational networks described here are composed of a collection of functional modules that take a certain collection of inputs and produce a collection of outputs. We define the notion of a time step to be the time it takes for a module of a network to compute its results from its inputs. In this manner, a time step can be viewed as the time quantum separating sequential sets of inputs to modules and outputs from modules.

The formalism we use to establish a correspondence between a mathematical expression and computational networks follows Cohen [3] by modeling a storage element (e.g., a flip-flop) with an operator that can be interpreted as a delay when acting on a series of conceptually sequenced data. A data sequence can be viewed such that successive elements of the sequence are separated by a single time step. In a self-timed system, the delay may vary in terms of absolute time, but the sequential order is always preserved. The operator is well defined mathematically and can be readily introduced in an expression to be evaluated by a network once the input data has been ordered in time. Hence, the mathematical expression can be transformed in a straightforward manner into a form that maps directly to a computational network. There are a number of mathematical expressions that are all functionally equivalent but whose direct hardware interpretation results in different networks. These equivalent forms can be obtained by formal manipulation of the equations. Correctness is assured since these transformations are function preserving.

Using this formalism it is possible to determine the essential properties of the networks directly from the equations defining them. Computational rate, performance, delay, modularity, and module count are all easily determined from the equations. The interconnection scheme and communication characteristics can also readily be found from the equations for networks with a high degree of regularity. We will show how this approach can be used to derive and characterize computational networks for Finite Impulse Response (FIR) filters, matrix-vector products, and the Discrete Fourier Transform (DFT). The details of these networks may be found in Cohen [3], Cohen and Tyree [4], Weiser and Davis [14] and Johnsson and Cohen [6] and [7]. Weiser and Davis [14] have also used this approach to treat networks for the multiplication of band matrices and the solution of triangular linear equations.

The mathematical approach pursued in this paper may also be used for verification. The modules used for the examples in this paper contain additions, multiplications, and delays. Their function can be described as a transfer function in the form of an operator that can be represented by a matrix. Since all of these networks are linear, the compositions of modules into arrays correspond in the functional domain to multiplication of matrices.

This mathematical approach is demonstrated in this paper only for one-dimensional arrays; however, this is not a limitation of the approach. See Weiser and Davis [14] for the application of this approach to two-dimensional arrays.

The progression of a computation can be modeled by the concept of a wave front. Wave fronts are an intuitively appealing way to illustrate how the computations proceed. Wave fronts can be defined either graphically, in terms of the networks, or mathematically, in terms of equations (see Johnsson and Cohen [6,7], and Weiser and Davis [14]). The use of wave fronts in the mathematical domain has the additional utility of simplifying the notational complexity. S. Y. Kung [9] has also used wave fronts in an informal way to describe the progression of computations in orthogonal arrays.

## Notation

Space and time are fundamental characteristics of computational networks and their behavior. Computations may be distributed in time, in space, or both.

Let  $X = \{x(k)\}$  be a sequence of variables. The index  $k$  is associated with time such that  $x(k)$  precedes  $x(k+1)$  by one time step. We refer to such a sequence as a data stream.

Define the operator  $Z$  by  $Zx(k) = x(k-1)$  and define  $Z^j = ZZ^{j-1}$ . Then  $Z^j x(k) = x(k-j)$ .

The elements of the sequence  $X$  may be observed in two fundamentally different ways. The first way is to view the elements over time as they pass by a particular point, which is fixed in space. The second way is to take a "snapshot" of the network, i.e., to view the elements of  $X$  as they are spread out in space at an instant of time. Such a snapshot is shown in Figure 1.

Hence, the operator  $Z$  may be considered as a delay in time when applied to a data stream at a certain position, or as a "shift over space" when considered for an instant of time.

Figure 1 shows the effect of  $Z^5$  operating on a data stream.



Figure 1:  $Z^5$  operating on a data stream

From the definition of  $Z$  and Figure 1 it is natural to interpret  $Z$  as a delay when acting on a data stream.

Define  $Z^{-1}$ , the inverse of the  $Z$  operator, by  $Z^{-1}x(k) = x(k+1)$ .

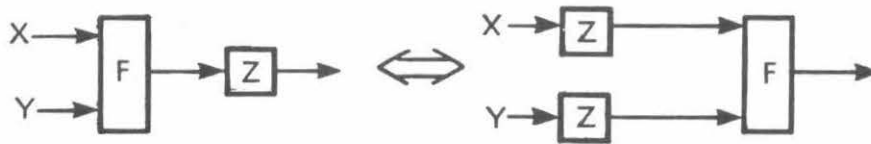
$Z^{-1}$  can be interpreted as a prediction when operating on data streams. It has the following properties

$$ZZ^{-1} = Z^{-1}Z = Z^0 = I, \text{ where } I \text{ is the identity operator.}$$

The operator  $Z$  is commutative with respect to time-independent functions. However, when commuting the  $Z$  operator and the function, the  $Z$  operator must be distributed over the entire operand set of the function, e.g.,

$$ZF(X,Y) = F(ZX,ZY).$$

A graphical representation of this commutative-distributive property is shown in Figure 2.

Figure 2:  $ZF(X,Y) = F(ZX,ZY)$ 

Constants do not change over time and the  $Z$  operator does not affect constants. Therefore,

$$Z(CX) = (ZC)(ZX) = C(ZX).$$

Hence, as operators, Z and C commute:  $(ZC)X = (CZ)X$ .

Equations in which the Z operator is used to express sequencing can be given a direct interpretation in terms of computational networks. Different expressions correspond to different networks. In a few examples we will show not only the correspondence between expressions using the Z operator and computational networks but also how some properties of the computational network (such as modularity, computational rate, performance, and fault propagation) can be determined directly from the expressions.

## Finite Impulse Response Filters

A Finite Impulse Response (FIR) filter can be defined as

$$y(k) = \sum_{i=0}^{N-1} a(i)x(k-i) \quad (1)$$

where X is the input signal to the filter and  $a(i)$ ,  $i = 0, 1, 2, \dots, N-1$  are the filter coefficients. The output of the filter is Y. The indices of X and Y are naturally associated with time in a real-time environment. The N multiplications required to compute each value of Y can be carried out in any order or concurrently, because equation (1) does not prescribe any order of evaluation. There exist several hardware realizations of equation (1). They may differ with respect to computational rate, performance, amount of hardware, reliability, etc.

For the FIR filter it is natural to assume that the data arrives sequentially and in order of increasing indices. Using the Z operator we will now discuss a few implementation alternatives.

Introducing the Z operator into equation (1) gives

$$y(k) = \sum_{i=0}^{N-1} a(i)Z^i x(k) \quad (2)$$

or

$$Y = \left( \sum_{i=0}^{N-1} a(i)Z^i \right) X$$

A direct hardware interpretation of equation (2) would contain  $N(N-1)/2$  delays,  $N$  multipliers, and  $N-1$  adders. Having computed the products, which can be made concurrently, the  $N$  terms have to be added. If the  $N-1$  additions are concurrent then the computational rate is limited by the carry propagation.

However, equation (2) can be rewritten as

$$y(k) = \left( \dots((a(N-1)Z + a(N-2))Z + a(N-3))Z + \dots Z + a(0) \right) x(k) \quad (3)$$

From equation (3) it is obvious that  $N-1$  delays suffice to implement the filter defined by equation (1). Equation (3) naturally corresponds to a linear array of modules, as indicated in Figure 3. The computational rate of an implementation corresponding to equation (3) is, however, still limited by  $N-1$  concurrent additions.

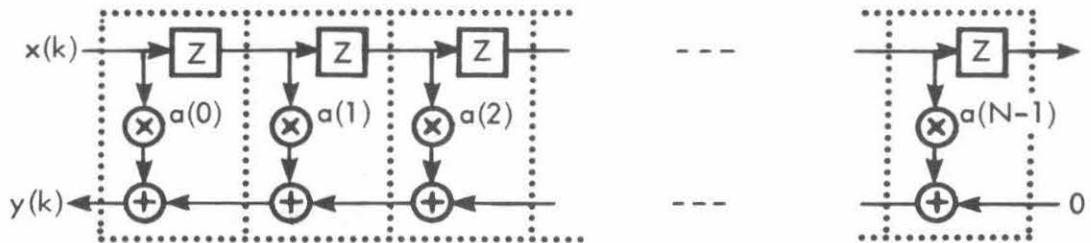


Figure 3: The implementation of the FIR filter

The coefficients  $a(i)$ ,  $i=0,1,2,\dots,N-1$ , and the  $Z$  operator commute because the coefficients are constants. Note that the index  $k$  is associated with time and that the index  $i$  is associated with space ("module number"). Therefore,  $Z$  operates on the variable  $k$ , not on  $i$ . Using the property that  $a(i)$  and  $Z$  commute, equation (2) can be rewritten as

$$y(k) = (a(0) + Z(a(1) + Z(\dots(a(N-2) + Z(a(N-1)))\dots))) x(k)$$

or

$$Y = \left( \sum_{i=0}^{N-1} Z^i a(i) \right) X. \quad (4)$$

Equation (4) corresponds to an implementation that has as many components as the implementation of equation (3), but the modules and the array have different properties.  $x$  is broadcasted to all modules. For large values of  $N$ , broadcasting ("fanout") is undesirable. Broadcasting implies long wires and large drivers; long wires are likely to be slow and may limit the computational rate. An implementation corresponding to equation (4) is not, however, limited by  $N-1$  concurrent additions, as is the implementation of equation (3). The summation in implementations corresponding to equation (4) performed in a pipelined fashion. Figure 4 shows modules and a linear array corresponding to equation (4).

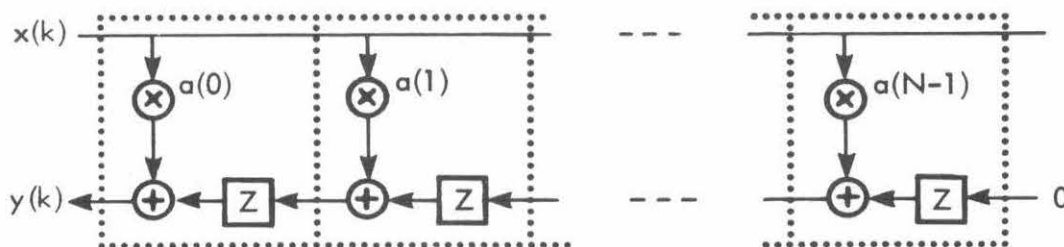


Figure 4: The implementation of the FIR filter

If the value of  $N$  is large enough for broadcasting to be a problem, then the computations of the FIR filter can be grouped so that broadcasting will not be a problem in each group. The groups are then connected via delays. The output of the FIR filter will be delayed a number of steps one less than the number of groups. The broadcasting may also be made in a tree-like manner. One possible solution to this fanout problem is shown in Figure 5.

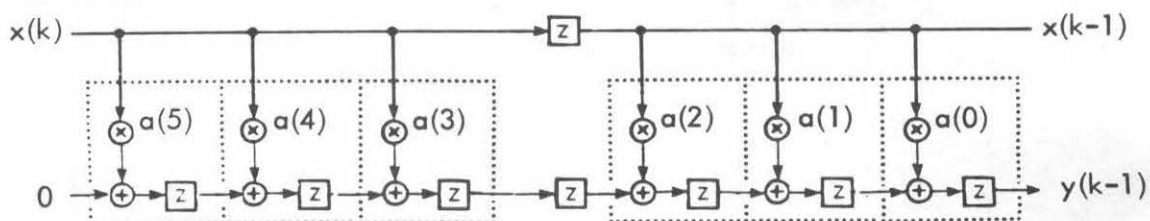


Figure 5: The implementation of the FIR filter

In the implementations discussed above, it has been assumed that  $N$  is small enough to allow for all modules required by the equations as given to be implemented. The input consisted of one data stream.

### Matrix vector multiplication

The product  $Y$  of a matrix  $A$  by a vector  $X$  is defined by

$$y(m) = \sum_{i=1}^N a(m,i)x(i) \quad \text{for } m = 1, 2, \dots, M \quad (5)$$

Hence, each  $y(m)$  is the inner product of the  $m$ th row of  $A$ ,  $\{a(m,*)\}$ , and the vector  $X$ .

The evaluation of each of these inner products is similar to the evaluation of the FIR filter shown above, with the differences that here there is a set of  $\{a(i,*)\}$  associated with the  $i$ th unit, not just one  $a(i)$  as before, and that the notation here starts at  $i=1$  whereas it starts at  $i=0$  in the FIR filter computation.

$M \times N$  multiplications are required for the evaluation of equation (5). In the following discussion they are distributed into  $M$  modules (distribution in space) each performing  $N$  multiplications (distribution in time).

Obviously, these  $M$  inner products are not independent because they share the same input vector,  $X$ .

We first introduce another implementation scheme for inner products; then we show several different ways to interconnect them in order to achieve the matrix-vector multiplication.

A straightforward use of the arrays discussed above for the FIR filter to compute a matrix-vector product would require one array for each component of the product, i.e.,  $O(N \times M)$  modules. Since this quantity may be prohibitively high, we use another approach that uses only  $M$  modules. This reduction in the number of modules is obviously reflected in the rate at which the output is computed, as seen below.

We follow approach B from Cohen and Tyree [4]. We pursue an implementation that is organized as  $M$  modules, each corresponding to a certain  $y(m)$ . The matrix coefficients are given one column at a time such that each row,  $\{a(m,*)\}$ , is a data stream of coefficients given to the unit corresponding to that  $y(m)$ . The vector  $X$  is also given as such a data stream to all the units.

With this organization, the operation of the operator  $Z$  on the data is

$$Zx(k) = x(k-1) \text{ and } Za(m,k) = a(m,k-1) \quad (6)$$

Note that the above is a property of the data organization and not of the  $Z$  operator.

Define the partial sums involved in the computation of the products:

$$Y(m,k) = \sum_{i=1}^k a(m,i)x(i) \text{ for } k = 1, 2, 3, \dots, N$$

Obviously,  $y(m) = Y(m,N)$ . Also,  $Y(m,0) = 0$ .

The partial sums are recursively computed by  $Y(m,k) = Y(m,k-1) + a(m,k)x(k)$ .

Hence,  $Y(m,k) = ZY(m,k) + a(m,k)x(k)$ , which can be written as  $(I-Z)Y(m,k) = a(m,k)x(k)$ .

Multiply both sides by  $(I-Z)^{-1}$  and get

$$Y(m,k) = (I-Z)^{-1}[a(m,k)x(k)] = \sum_{i=0}^{\infty} Z^i[a(m,k)x(k)] = \sum_{i=0}^{\infty} a(m,k-i)x(k-i) \quad (7)$$

A module for the implementation of equation (7) is shown in Figure 6.

However, it is apparent that a module corresponding to equation (7) has an infinite "response". The boundary conditions of the problem imply a need to bound this infinite response. This is done by using a modulo  $N$  counter to provide a reset mechanism, as in Cohen and Tyree [4].

With this additional control, designed for repetitive operation, the module can be redefined as

$$Y(m,k) = ZY(m,k) + a(m,k)x(k) \quad \text{for } k \neq 1 \pmod{N}$$

and

$$Y(m,k) = 0 + a(m,k)x(k) \quad \text{for } k = 1 \pmod{N}. \quad (8)$$

A module for the implementation of equation (8) is shown in Figure 7.

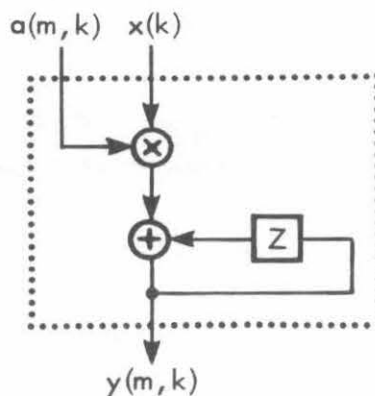


Figure 6: Infinite response module

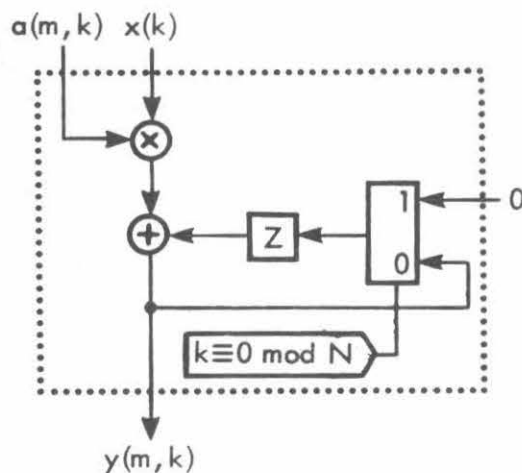


Figure 7: A finite response module

The reset occurs at the same time that the partial sum,  $Y(m, k)$ , is equal to  $y(m)$ . At this time  $y(m)$  is output, and the computation of the next  $y(m)$  begins.

Note that here there is a need for control signaling, which was not needed in the FIR case. This is because of the distribution of the FIR computation in space, where the "size" of the inner product ( $N$ ) is determined by the actual size of the array, which implicitly defines the value of  $N$ . In the latter case

this computation is distributed in time, which necessitates the use of control signals for defining the value of  $N$ .

The discussion above focused on the single module for inner products. There are several ways for using  $M$  such devices in the design of a network for matrix-vector multiplication.

One way is to synchronize  $M$  such devices in parallel and to supply the same  $x(k)$  to all of them at the same time through any of many broadcasting techniques.<sup>1</sup> In such an arrangement a single reset control signal is broadcasted to every unit, and all the values of  $\{y(m), m = 1, M\}$  are available at the same step. Such a network is shown in Figure 8.

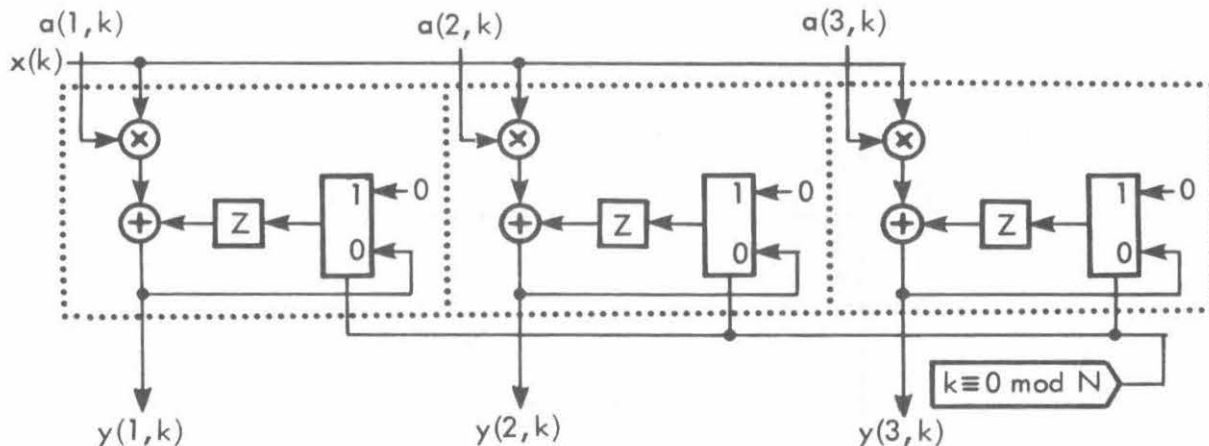


Figure 8: Synchronized matrix-vector multiplication

Another possible arrangement is based on the following relation:

$$Z^m y(m) = \sum_{i=1}^N Z^m [a(m, k) x(k)] = \sum_{i=1}^N [Z^m a(m, k)] [Z^m x(k)] = \sum_{i=1}^N a'(m, k) [Z(m) x(k)],$$

where  $a'(m, k) = a(m, k-m)$ , namely the same sequence "shifted" in time by  $m$  steps.

<sup>1</sup> It is assumed that the broadcasting delivers the same value to all of its recipients "at the same time".



This network has one delay in  $X$  between successive modules. It also changes the "phase" of each unit, such that the  $m$ th unit has to be reset when  $k = m \pmod{N}$ , unlike the condition  $k = 0 \pmod{N}$  as above. Hence, each unit is reset at a different time (rather than all of them at once) and the elements of  $Y$  are available sequentially. In this implementation the input stream is continuous, one  $x(k)$  per cycle, as is the output stream, one  $y(m)$  per cycle.

The network for implementing this arrangement is shown in Figure 9.

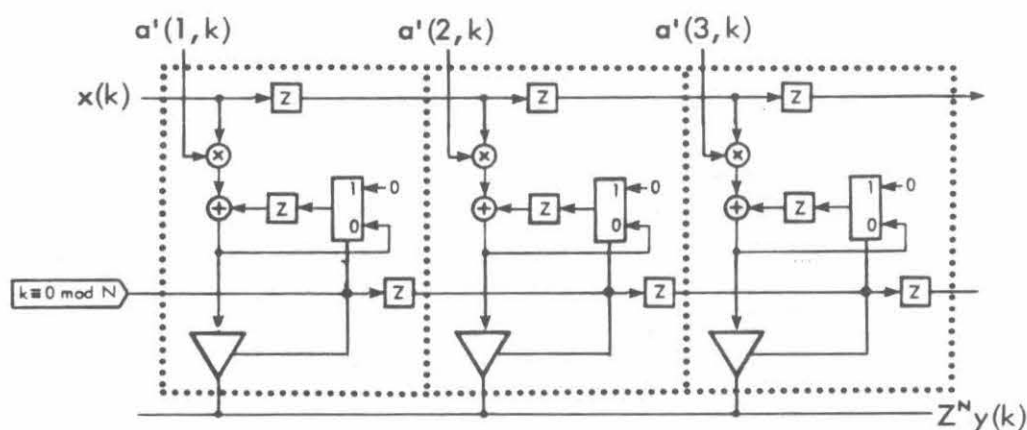


Figure 9: Pipelined matrix-vector multiplication

According to Figure 9, an array of  $M$  modules is capable of concurrently computing sequences of matrix-vector products of indefinite length,  $N$ .

Another related problem is the multiplication of a band matrix by a vector. A matrix  $A$  is defined to be a band matrix of width  $r + s + 1$  if  $a(i, j) = 0$  for all  $j > i + r$  and for all  $i > j + s$ .

A FIR filter can be considered as a multiplication of a band matrix by a vector. The filter coefficients are the diagonals of the matrix.

The structure of a band matrix suggests that the elements of the matrix are entered along its diagonals such that each diagonal is entered as a separate data stream.

In this arrangement the effect of  $Z$  on the matrix elements is  $Za(m,k) = a(m-1,k-1)$ ,

The difference between this expression and equation (6) is due to the different order of the input data.

The concept of a wave front is useful because it simplifies the mathematical notation and it can be used as a conceptual tool in understanding how data progresses through a computational network. Wave fronts in computational networks are analogous to wave fronts in fluid dynamics. They are of particular importance in investigating laminar flow but are less useful in the study of turbulent flows.

Elements of a sequence of data values ordered in time can be viewed as a data stream. Elements of the same data stream are separated by a single time step and typically follow the same path through a computational network.

A wave front in fluid dynamics consists of a set of points in space which changes with time according to the propagation of the wave. Similarly, a wave front in a computational network consists of elements from different data streams. If we view a computational network abstractly as computing a result based on input operands, then it is possible to associate a set of data streams with a particular input operand. In the previous example of matrix-vector multiplication, the matrix operand consisted of a set of data streams, where each data stream corresponded to a row of the matrix. Using the wave front concept it is possible to change the view of an operand from a set of data streams to a set of wave fronts. These wave fronts are essentially a series of parallel cross sections of the set of data streams. More precisely, such a wave front contains exactly one element from each data stream. We are particularly interested in wave front sequences that contain all of the data elements present in the set of data streams comprising the operand.

In the implementation of the matrix-vector multiplication, as shown in Figure 8, all of the multiplications corresponding to a column are performed concurrently, and the matrix elements enter the array column by column. The input data set applied at time  $k$  may be defined as a wave front  $WF(k)$ . Applying the  $Z$  operator to every data stream results in the wave front  $WF(k-1)$ .

In general,  $ZWF(k) = WF(k-1)$ . By this definition a wave front corresponds to a column of the matrix  $A$ . Figure 10 shows these wave fronts.

However, in the implementation shown in Figure 9, the wave fronts are skewed because of the relative delay between successive modules. These wave fronts are shown in Figure 11.

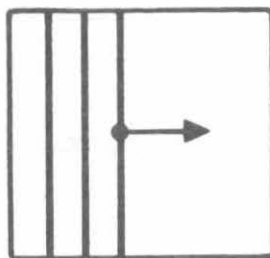


Figure 10: Vertical wave fronts

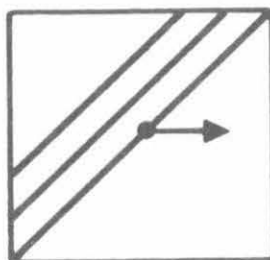


Figure 11: Skewed wave fronts

It is possible to perform several types of transformations on wave fronts. For example, it is possible to transform a wave front representing a row in a band matrix to a wave front representing a column in that matrix, and vice versa. This is done by applying  $Z^i$  to each element of the initial wave front, where the value of  $i$  corresponds to the data element position in the initial wave front. This is, in effect, a rotation of the initial wave front that results from applying the variable delay to its elements.

## The Discrete Fourier Transform

The Discrete Fourier Transform (DFT) is defined by

$$y(k) = \sum_{m=0}^{N-1} w^{mk} x(m) \quad \text{for } k = 0, 1, 2, \dots, N-1$$

where  $w = e^{-2\pi i/N}$

The DFT can be considered as a special case of a matrix-vector product. The approach corresponding to equation (8) is directly applicable. The elements in a row or a column can be easily generated because the ratio between consecutive elements is a constant. The computational

networks we derive here explore the fact that the ratio between consecutive elements in a column is a constant.

Define  $Y(m,k) = w^{mk}x(m)$  such that each  $Y(m,*)$  is a data stream. Therefore,  $ZY(m,k) = Y(m,k-1)$ .

The sequence  $\{Y(m,k), m = 0, 1, 2, \dots, N-1\}$  may be generated by

$$Y(m,k) = w^m Y(m,k-1) = w^m ZY(m,k) = w^{mk} Y(m,0)$$

and  $Y(m,0) = x(m)$ .

Obviously,

$$y(k) = \sum_{m=0}^{N-1} Y(m,k) \text{ for } k = 0, 1, 2, \dots, N-1. \quad (9)$$

A module generating the variables  $Y(m,*)$  is shown in Figure 12.

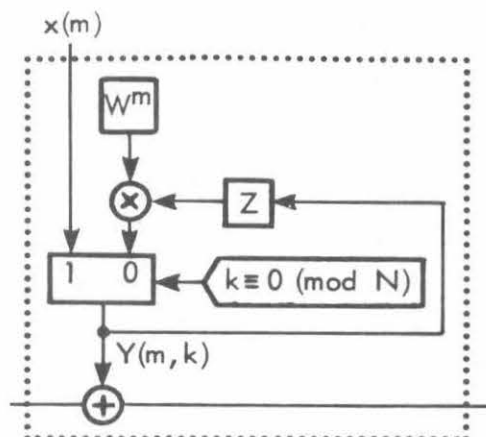


Figure 12: The  $Y(m,*)$  module

An array corresponding to equation (9) suffers from the need to perform  $N-1$  additions in one step. The modules in the array are initiated with different values,  $\{x(m)\}$ , which become available at successive steps. The input values therefore have to be stored for the initialization of the modules.

The implementation corresponding to equation (9) can be improved by using pipelined addition.

Let the  $\{Y(m,*)\}$  modules be interconnected into an array in a pipelined manner as shown in Figure 4.

Let  $s(k)$  be the output of this array at time  $k$ ; then

$$s(k) = Y(N-1,k) + Z(Y(N-2,k) + Z(\dots Z(Y(0,k))\dots)). \quad (10)$$

The modules are initialized so that

$$\begin{aligned} Y(m,k) &= x(k) && \text{for } k = m \pmod{N}, \quad m = 0, 1, 2, \dots, N-1 \\ \text{and} \\ Y(m,k) &= w^m Y(m, k-1) && \text{otherwise.} \end{aligned} \quad (11)$$

Equation (11) expresses a sampling mechanism. The values of the data stream  $X$  are multiplexed into the  $N$  modules in a cyclic manner. The output  $S$  is well defined from time  $N-1$  and on. To study the output  $S$  we rewrite equation (10) as

$$s(k) = Y(N-1,k) + Y(N-2,k-1) + Y(N-3,k-2) + \dots + Y(0,k-(N-1)),$$

$$\text{Hence } s(N-1) = x(N-1) + x(N-2) + \dots + x(0) = y(0)$$

$$s(N) = w^{(N-1)}x(N-1) + w^{(N-2)}x(N-2) + \dots + w^0x(0) = y(1)$$

...

$$s(2(N-1)) = w^{(N-1)(N-1)}x(N-1) + w^{(N-2)(N-1)}x(N-2) + \dots + w^{0(N-1)}x(0) = y(N-1)$$

$$s(2N-1) = x(2N-1) + x(2N-2) + \dots + x(N) = y(N)$$

...

An array corresponding to equations (10) and (11) is shown in Figure 13.

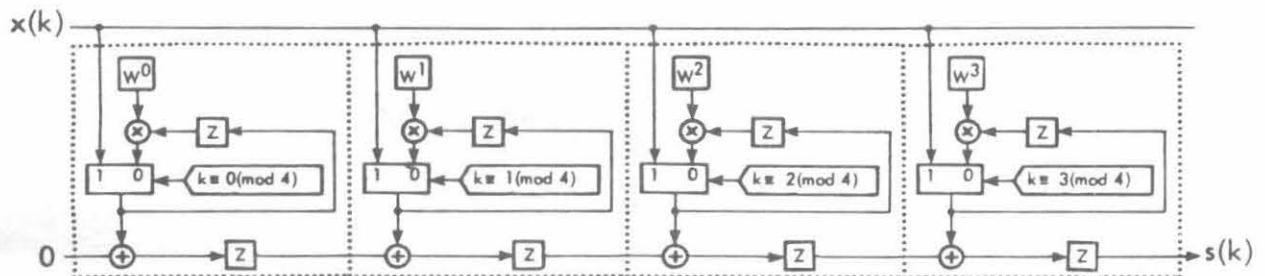


Figure 13: DFT array of size 4

The set of multiplications being performed during a step of the computations lies on lines parallel to the diagonal from the lower left-hand corner to the upper right-hand corner of the matrix defining the DFT. These lines can be defined as wave fronts. The computations start at the upper left-hand corner and proceed downward and to the right. The wave fronts are illustrated in Figure 14. The multiple wave fronts shown in Figure 14 correspond to the case where a sequence of DFTs are computed by the array. There is one wave front for each DFT. Hence, computations belonging to two DFTs are typically performed concurrently.

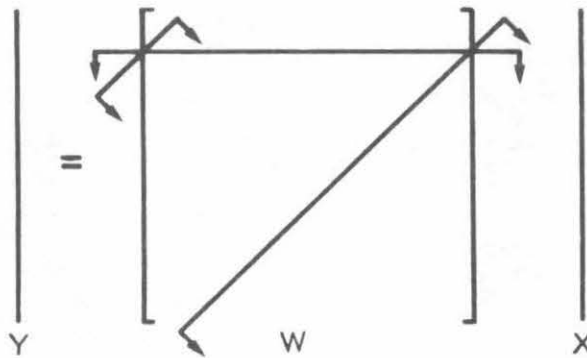


Figure 14: Wave fronts for DFT

Multiplexing the elements of the stream into a set of  $N$  modules can also be used in the general matrix-vector multiplication case. The matrix elements cannot in general be generated within a module but have to be supplied to the modules. Each module should be supplied with the matrix elements in a column in row order. The data stream associated with a column should be delayed one time step with respect to the stream associated with the preceding column. The loop around the multiplier is replaced with a storage element into which the elements of the stream  $X$  are multiplexed. The output of the multipliers should be added in a pipelined manner. Figure 15 illustrates the data organization schematically with the matrix rows indicated by dashed lines. Wave fronts can be associated with the set of matrix elements that enters the array at any given time. The wave fronts defined in this way are indicated by solid lines in Figure 15 and correspond to the diagonals in Figure 14.

The implementation corresponding to equations (10) and (11) contains  $O(N)$  modules and computes the DFT in  $O(N)$  time. Following the FFT logic it is possible to reduce the number of modules to  $O(\log_2 N)$ , by further exploration of the properties of the coefficients.

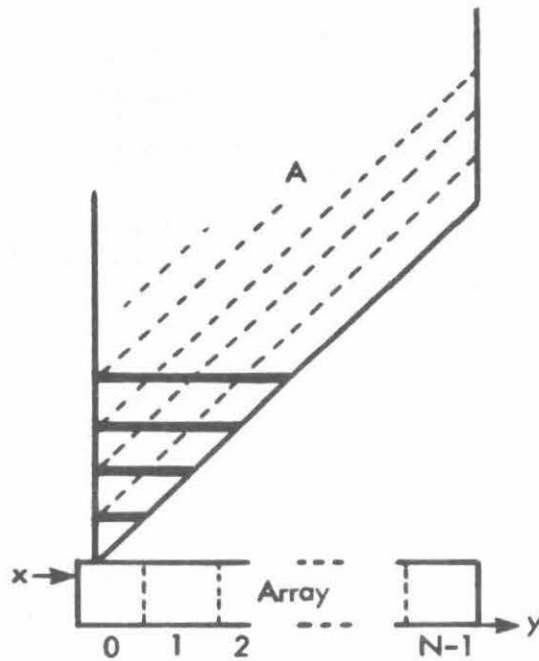


Figure 15: Wave fronts for DFT, v2

We assume  $N = 2^n$ .

The coefficient  $\{w^{jk}\}$  has the property

$$w^{(j + N/2)k} = \begin{cases} = w^{jk}, & k \text{ even}, j = 0, 1, 2, \dots, N/2-1 \\ = -w^{jk}, & k \text{ odd}, j = 0, 1, 2, \dots, N/2-1. \end{cases}$$

Hence,

$$y(2j) = \sum_{k=0}^{N/2-1} w^{2jk} (x(k) + x(k + \frac{N}{2})) \text{ for } j = 0, 1, 2, \dots, \frac{N}{2}-1$$

and

$$y(2j+1) = \sum_{k=0}^{N/2-1} w^{k(2j+1)} (x(k) - x(k + \frac{N}{2})) \text{ for } j = 0, 1, 2, \dots, \frac{N}{2}-1$$

or

$$y(2j+1) = \sum_{k=0}^{N/2-1} w^{2jk} (w^k (x(k) - x(k + \frac{N}{2}))) \text{ for } j = 0, 1, 2, \dots, \frac{N}{2}-1.$$

Thus, the even and odd components of the DFT can be obtained as matrix-vector products by using the same  $N/2$  by  $N/2$  matrix operating on different vectors.

Define a new data stream  $V$  obtained from the stream  $X$  as follows:

Let 
$$v(k) = (1 + Z^{-N/2})x(k) \text{ for } k = 0, 1, 2, \dots, N/2$$

and 
$$v(k) = w^k(1 - Z^{N/2})x(k) \text{ for } k = N/2, N/2 + 1, \dots, N-1.$$

The former definition requires a negative power of  $Z$ , a prediction that cannot be implemented in general. However, by multiplying both sides of this definition by  $Z^{N/2}$  no prediction is needed. This means that instead of computing  $v(k)$ , the network actually computes  $Z^{N/2}v(k)$ , which is the desired value delayed by  $N/2$  steps. Since the first half of  $\{v(k)\}$  has to be delayed by  $N/2$  steps we also delay the second half by the same amount.

A module computing the sequence  $V$  is shown in Figure 16.

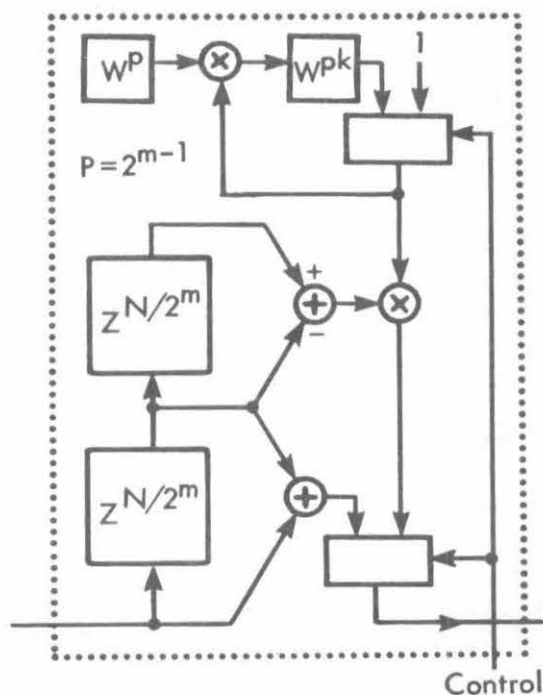


Figure 16: A modified butterfly module



Then

$$y(2j) = \sum_{k=0}^{N/2-1} w^{2jk} v(k) \quad \text{for } j = 0, 1, 2, \dots, \frac{N}{2}-1$$

and

$$y(2j+1) = \sum_{k=N/2}^{N-1} w^{2jk} v(k) \quad \text{for } j = 0, 1, 2, \dots, \frac{N}{2}-1$$

(12)

or in matrix form

$$\begin{bmatrix} y(0) \\ y(2) \\ \dots \\ y(N-2) \\ y(1) \\ y(3) \\ \dots \\ y(N-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 & 0 & 0 & 0 & \dots & 0 \\ 1 & w^2 & w^4 & \dots & w^{N-2} & 0 & 0 & 0 & \dots & 0 \\ 1 & \dots & \dots & \dots & \dots & 0 & 0 & 0 & \dots & 0 \\ 1 & \dots & \dots & \dots & \dots & 0 & 0 & 0 & \dots & 0 \\ 0 & w & w^3 & \dots & w^{N-1} & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & \dots & 0 & 1 & 1 & 1 & \dots & 1 \\ 0 & 0 & 0 & \dots & 0 & 1 & w^2 & w^4 & \dots & w^{N-2} \\ 0 & 0 & 0 & \dots & 0 & 1 & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0 & 1 & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 0 & 1 & w & w^3 & \dots & w^{N-1} \end{bmatrix} \begin{bmatrix} v(0) \\ v(1) \\ \dots \\ v(N/2) \\ \dots \\ v(N-1) \end{bmatrix}$$

The original matrix has been transformed into a block diagonal matrix with identical diagonal blocks. Obviously, the observation used to obtain equation (12) can be applied to each of the diagonal blocks recursively. Eventually, the block diagonal matrix becomes a diagonal matrix, and the computation of the DFT is completed. Figure 17 shows an array of  $\log_2 N$  modules computing the DFT.

The input has to be in normal order. The output appears in bit-reversed order. There is no broadcasting, and no module has to contain any long wires. The computational rate is limited by  $\log_2 N$  adders in series. The computational rate can be improved by introducing delays in a straightforward manner. The first component of the DFT will appear at the output of the array at the time the last data item,  $x(N-1)$ , arrives, and the last component will appear at the output  $N-1$  steps later.

Please note that this implementation performs FFT by using decimation in frequency.

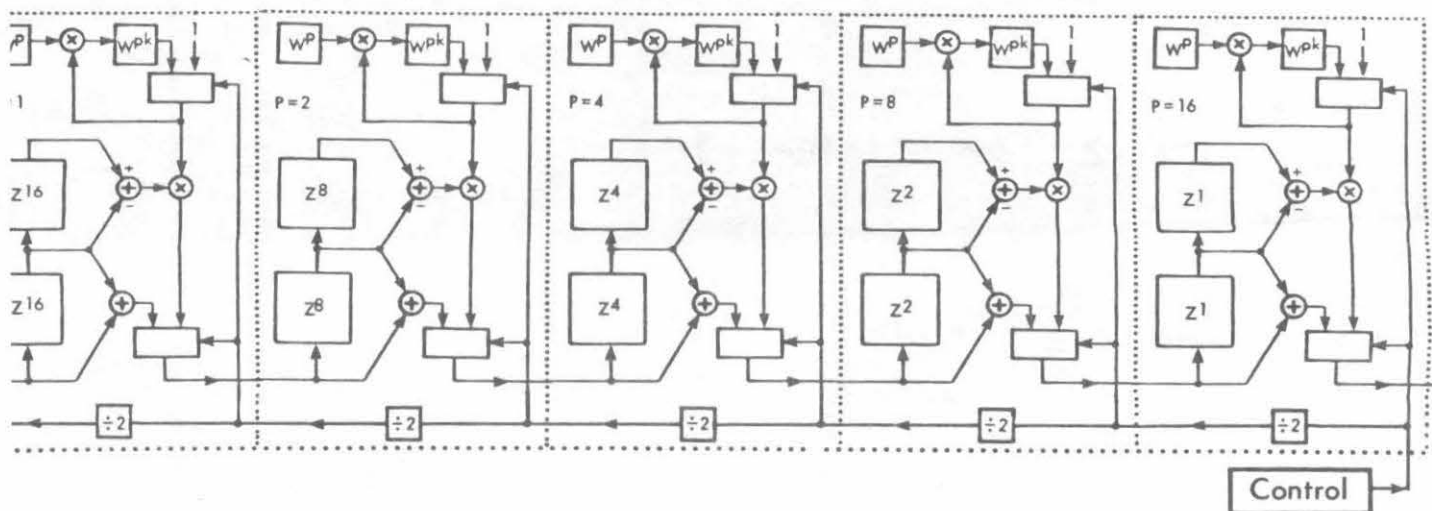


Figure 17: An FFT array

In the FIR filter, the matrix-vector multiplication cases, and the DFT network of  $N$  modules, the data streams can be considered as laminar. The wave fronts can be considered as corresponding to points of constant phase on waves. The network considered for the computation of the DFT by  $\log_2 N$  modules does not preserve the laminar flow. The concept of wave fronts is not attractive in this case as in the case for turbulent flows.

## Conclusions

The simple model of storage used in this paper is given a precise mathematical meaning. Once the organization of the input data is determined, the  $Z$  operator (i.e., the mathematical model of a storage element) can be used to model the ordering of input data streams directly. The mathematical equation that defines a function to be computed can be transformed from a form that contains no concept of time to a form that contains information about time as well as space. This new form is typically an expression containing the  $Z$  operator.

Equations containing information about the time and space required to compute a function can be manipulated formally, since the properties of the  $Z$  operator are well defined. It is possible to give expressions containing the  $Z$  operator a direct hardware interpretation. Properties such as computational rate, performance, delay, modularity, communication structure, and fault tolerance can be determined directly from an expression using the  $Z$  operator.

The methodology suggested in this paper is useful in synthesis as well as analysis and verification of computational networks. It is possible to iterate between formal manipulations of expressions in the  $Z$

operator and graphs describing computational networks. Modeling the behavior of a network using the Z operator to describe a storage element makes it conceptually straightforward to verify whether or not the network actually computes the function it is supposed to compute.

A formal approach as outlined in this paper is particularly useful in complex problems where designs based entirely on intuition may be incorrect or may have a performance lower than necessary for a given amount of hardware. For instance, using the methodology suggested here, Weiser and Davis [14] have discovered designs of systolic arrays with two to three times higher performance than the corresponding arrays by H. T. Kung and Leiserson [8]. Furthermore, the designs are proven to be correct.

Explicit control can be modeled within the formalism. The expressions become more complex, but the formalism allows for the treatment of space-time tradeoffs. If at the first iteration of the design cycle the spatial requirements of a computational network for a function are too large, the hardware requirements can be reduced by mapping the computations to a network of reasonable size. In doing so, it is necessary to model the control explicitly. Eventually the control will become fairly complex, but it can still be included in the formalism.

The formalism allows for a precise definition of wave fronts. The concept of wave fronts is useful both in designing computational networks and in finding suitable organizations of the input data. Wave fronts are particularly useful in problems where the data flow can be considered laminar. The formalism proposed is not, however, limited to networks with laminar flow as the example describing arrays for the FFT shows.

## Acknowledgments

The authors gratefully acknowledge the support for this research provided generously by the Defense Advanced Research Projects Agency, under contracts MDA-80-C-0523 with the USC/Information Sciences Institute and N00014-79-C-0597 with the California Institute of Technology, and by the Burroughs Corporation for the Data Driven Research Project at the University of Utah.

Views and conclusions contained in this paper are the authors' and should not be interpreted as representing the official opinion or policy of DARPA, the U.S. Government, any person or agency connected with them, nor of the Burroughs Corporation.

## References

1. Ayres, R., "IC design language," *Proceedings of the 16th Design Automation Conference*, June 25-27, 1979, pp. 307-309. IEEE No. 79CH1427-4, Library of Congress Card No. 76-150348.
2. Chen, M., and C. Mead, "A notation for designing concurrent systems," *Internal Document (3927)*, Caltech, Computer Science Department, August 1980.
3. Cohen, D., "Mathematical approach to iterative computational networks," *Proceedings of the Fourth Symposium on Computer Arithmetic*, pp. 226-238, October 1978, also published as USC/Information Sciences Institute RR-78-73, November 1978.
4. Cohen, D., and V. C. Tyree, "VLSI system for Synthetic Aperture Radar (SAR) processing," *Proceedings of the Society of Photo-Optical Instrumentation Engineers (SPIE)*, Vol. 186, pp. 166-177, 1979.
5. Johannsen, D., "Bristle Blocks: A silicon compiler," *Proceedings of the Caltech Conference on VLSI*, January 1979.
6. Johnsson, L., and D. Cohen, "Computational arrays for the Discrete Fourier Transform," *COMPCON*, February 1981.
7. Johnsson, L., and D. Cohen, *A Mathematical Approach to Computational Networks for the Discrete Fourier Transform*, USC/Information Science Institute, RR-81-90, 1981 (forthcoming).
8. Kung, H. T., and C. E. Leiserson, "Algorithms for VLSI processor arrays," Section 8.3 in [10].
9. Kung, S. Y., "VLSI matrix computation array processor," *The MIT Conference on Advanced Research in Integrated Circuits*, February 1980.
10. Mead, C., and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980.
11. Rem, M., and C. Mead, "A notation for designing restoring logic circuitry in CMOS," *Second Caltech Conference on VLSI*, January 1981.
12. Rowson, J., *Understanding Hierarchical Design*, Caltech, Computer Science Department, Report 3710, April 1980.
13. Seitz, C., "System timing," Chapter 7 in [10].
14. Weiser, U., and A. Davis, *Mathematical Representation for VLSI Arrays*, University of Utah, Computer Science Department, Report UUCS-80-111, September 1980.

## A NOTATION FOR DESIGNING RESTORING LOGIC CIRCUITRY IN CMOS

Martin Rem

Eindhoven University of Technology  
and California Institute of Technology  
and

Carver Mead

Professor of Computer Science, Electrical Engineering  
and Applied Physics  
California Institute of Technology

## 1. INTRODUCTION

As the underlying silicon fabrication technology has become capable of producing chips with transistor counts in excess of 1,000,000, problems associated with correct design are assuming ever greater importance. Exhaustive checking of mask artwork for errors becomes prohibitive. Technologies and design styles which obviate large classes of potential errors are enormously preferable to those that do not.

A modular, hierarchical design style can, with proper restriction, confine many types of checks to one level of the hierarchy within each module. A set of such restrictions is given in this paper, together with a mechanism for their enforcement. These restrictions capture a substantial fraction of the design style given in [1].

As feature sizes are scaled below one micron, ratio logic processes like nMOS and I<sup>2</sup>L become progressively less attractive. Straightforward scaling to smaller sizes results in a linear increase in current per unit chip area. Technological tricks such as high resistivity polysilicon pullup devices or very small injector current can be used to decrease current drain, but the resulting devices become increasingly vulnerable to "soft error" problems from alpha particles, etc. Fully restored "static" logic using a complementary process is the natural choice for systems with submicron components. Present bulk CMOS processes have a number of very ugly analog rules associated with the 4-layer nature of the process. As a result, the designer must be aware of details of the technology to an alarming degree. CMOS on an insulating substrate is, on the other hand, a conceptually clean process: it requires no analog rules whatsoever if proper timing conventions are observed. There are recent signs that it may become reliably producible as well.

We introduce a programming notation in which every syntactically correct program specifies a restoring logic component, i.e., a component whose outputs are permanently connected, via "not too many" transistors, to the power supply. It is shown how the specified components can be translated into transistor diagrams for CMOS integrated circuits. As these components are designed as strict hierarchies, it is hoped that the translation of the transistor diagrams into layouts for integrated circuits can be accomplished mechanically.

In this paper we do not address the dynamic behavior of the logic components. The "proper timing conventions," alluded to above, are left for a subsequent paper.

## 2. SWITCHES IN CMOS

The CMOS technology uses two types of transistors: the N-channel enhancement transistor (1a) and the P-channel enhancement transistor (1b).

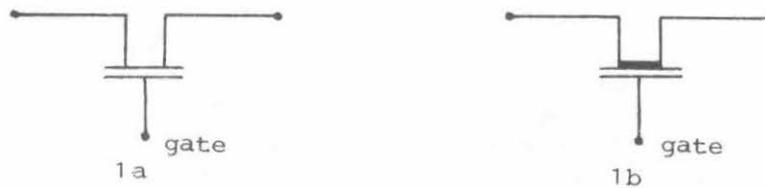


Fig. 1

Both of them act as switches but they are "on" and "off" for complementary values on their gates. Denoting a high voltage by "1" and a low voltage by "0", switch 1a is on if the gate is 1 and 1b is on if the gate is 0. When the switches are on, however, they do not convey a 1 and a 0 on their paths (in Fig. 1 the horizontal connections) equally well. Switch 1a conveys a 0 virtually perfectly, but it is not a perfect switch for a 1. Switch 1b, conversely, is a good conveyor for a 1 only.

Using these CMOS transistors we want to make two types of switches, a "normally-off" switch (2a) and a "normally-on" switch (2b).



Fig. 2

If the gate is 0 switch 2a is off (nonconveying) and 2b is on (conveying). Otherwise 2a is on and 2b is off. The points e1 and e2 are called the end points of the switch. We call the connection between the end points its path. If nothing is known about the values conveyed through its path, except that they are 0's and 1's, the realization of a switch requires two transistors: (the complement of  $g$  is denoted as  $g'$ )

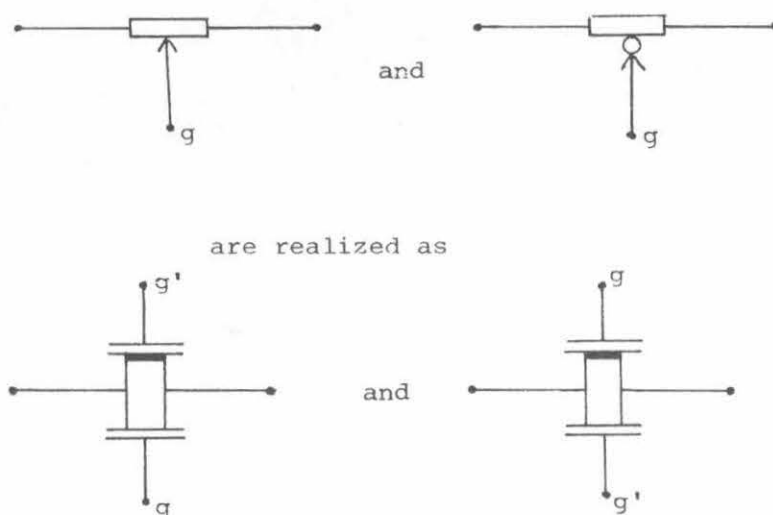


Fig. 3

These double transistors make our switches good conveyors for both 0's and 1's, which allows the use of longer strings of switches. These strings of switches, however, should not be too long: the distance to the "power supply" must not be excessive, otherwise the signal will become inaccurate and the circuit slow. To do justice to the nature of restoring logic we disallow the driving of external outputs by long strings of switches. This shall be reflected in the composition rules to be formulated in Section 3.

The gate inputs are run in two-rail logic to accommodate both the  $g$  and the  $g'$  signals. For switches that are known to convey always the same value there are two instances in which they can be realized by just one transistor:

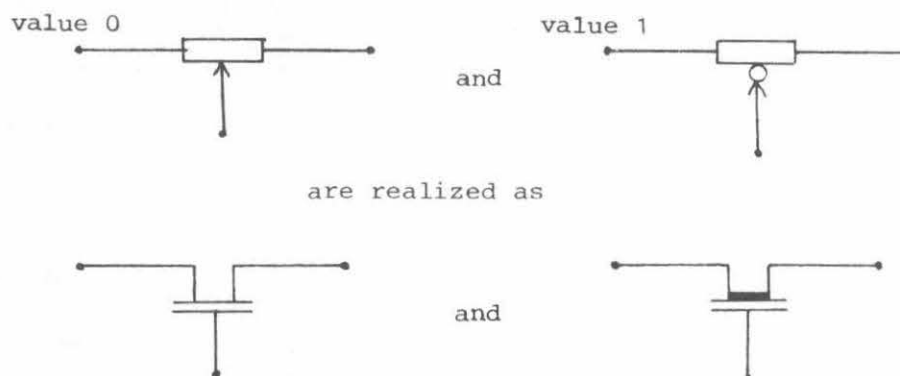


Fig. 4

In that case, the two-rail representation of the gate signal is not necessary. It is assumed that the compiler can recognize instances in which one transistor suffices. From now on we shall simply design in terms of switches and apply the above knowledge only if we wish to count the number of transistors a component requires.



### 3. RESTORING LOGIC COMPONENTS

A restoring logic component (RL) has external ports. The purpose of an RL is to establish a relation between the values it communicates via its external ports. We restrict ourselves to the values 0 and 1.

We design components in a hierarchical fashion. A typical RL is shown in Fig. 5.

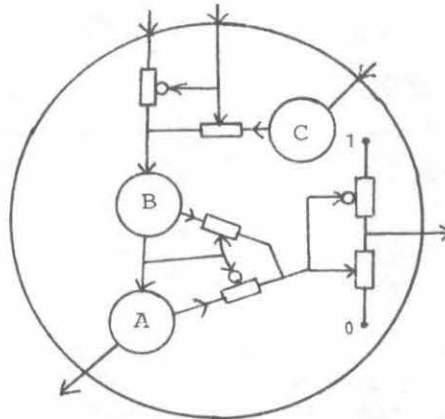


Fig. 5

It consists of subcomponents A, B, and C, which are also RL's, and a pattern of connections between them. We restrict the possible connection patterns to guarantee that the composite is again an RL. Such restrictions are only useful if they can be formulated in terms of the connection pattern, i.e., independent of the internal structures of the subcomponents thus connected. Before we can formulate these connection rules we have to give a few definitions. Each port is either an input port or an output port. The connection pattern of an RL specifies connections between its external ports and the external ports of the subRL's. We call the external ports of a subRL internal ports of the RL. An external output port of a subRL is an internal input port of the RL. Conversely every external input port of a subRL gives the RL an internal output port. The rules on connection patterns will be stated in terms of external and internal ports of the RL.

We assume that the distribution of power and ground to all components is taken care of by the compiler. Johannsen [1] has outlined a method for the distribution of power and ground over hierarchically defined components. In our nomenclature: each RL has two constant internal input ports, denoted by 0 and 1. These constants are the power supply rails which must be present in every component.

In Section 2 we have introduced the term path for the connection between the two end points of a switch. We now generalize that term. We say that there is a path between two ports p1 and p2 if either they are connected by a wire (a "wire path") or there is a switch such that there are paths between p1 and one end point of the switch and between p2 and the other end point. In the latter case we say that the switch is on the path. A path is called a conveying path if all switches on



the path are on. The values on the input ports (external or internal) determine which switches are on and which are off, and hence between which ports there are conveying paths. (Whenever we do not specify whether a port is external or internal, that is done intentionally.)

Two input ports are said to be fighting if there exists any assignment of values to all input ports such that there is a conveying path between the two input ports.

We introduce three rules the connection pattern must satisfy:

Rule 1. [no fighting]: No two input ports are fighting.

Rule 2. [restored external outputs]: Every external output port  
(a) has a wire path to an internal port, or  
(b) has a conveying path to 0 or 1 for every assignment of values to all input ports.

Rule 3. [nonfloating internal outputs]: For every internal output port  $p$  and for every assignment of values to all input ports there is a conveying path between  $p$  and an input port.

Notice that Rule 1 includes 0 and 1 (the two constant internal input ports). Remember that internal outputs are regarded as (external) inputs of the subcomponent and that the subcomponent's external outputs are internal inputs for the component.

The justification of Rule 1 is obvious. The result of Rule 2 is that all external outputs are driven by power or ground. They may be driven via a number of switches, but such a string of switches is confined to one component, viz. the component in which the actual connection to 0 or 1 is made.

The rules for internal outputs, i.e., outputs to subcomponents, are more liberal. We allow that inputs from subcomponents and inputs from the environment are directed through switches before they are output to subcomponents. For inputs from subcomponents this is reasonable: they are restored by the subcomponents. With inputs from the environment we have to be more careful. We have to allow that such a signal from an external input port goes through a switch to an internal output port. Otherwise we would be unable to make the flip-flop to be shown in Example 3. But it does allow long strings of switches "going into" the hierarchy, as sketched in Fig. 6.

We do not consider this a serious drawback. One may expect a subcomponent to have (physically) shorter connections than the component itself. Restoring in the "inward" direction, therefore, seems less vital than in the "outward" direction. Still, if we wish to bound the lengths of such inward strings of switches we could have the compiler insert amplifiers into them to restore their signals.

The consequence of allowing the switches in the outputs to subcomponents is that Rule 2 has to be stronger than one might expect. In Rule 2 we could not allow wire paths between external input ports and external output ports. This may seem to disallow running through a

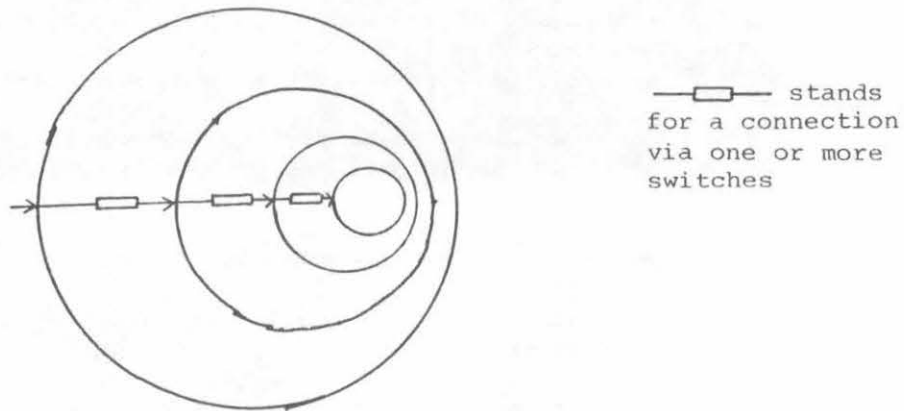


Fig. 6.

component wire whose signals are not used by the component. In fact, it does not. Such a wire is just not part of the component. (On the chip a wire between two components may run through the "area" of another component, but that is a matter of chip layout. It is a physical property, not a functional one.) Allowing wire paths between external input ports and external output ports would have given rise to the possibility of ill-restored outputs. Fig. 7 sketches an RL that is allowed by Rules 2 and 3. Now assume that each  $S_i$  is just a wire path from its input to its output, which would be allowed if we weakened Rule 2. The output of the RL is then not restored. Imagine now that each  $S_i$  actually has the same structure as the whole RL. It is clear that this would violate our goal of having restored external outputs.

In one respect is Rule 3 stronger than necessary. It requires that all subcomponents receive well-defined inputs, even a subcomponent whose outputs are not used. We could have restricted the rule to subcomponents whose outputs are actually used in the computation, but that would have made both the rule and the checking whether it is obeyed more complicated.

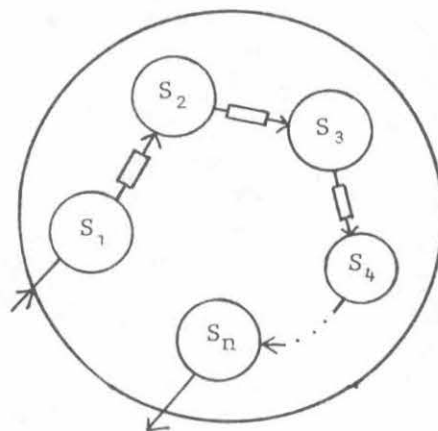


Fig. 7

## 4. THE PROGRAMMING NOTATION

In this section we introduce a programming notation in which connection patterns can be specified that satisfy the three rules of the preceding section. There are two properties a good notation should enjoy. First, it should be relatively simple for the compiler to check that a program is syntactically correct. If this mechanical check is simple, it will probably be simple for programmers to convince themselves that their designs satisfy the rules. We shall show how the syntactic checking can be performed. Second, it should be possible to give a formal definition of the semantics of our programs. We have not yet achieved the second goal, but ultimately we must be able to prove that a component performs a certain computation. That seems a much better technique than a demonstration of its effect with an a posteriori simulation. (Besides, how do we know that the simulation is correct if we do not have a rigorous definition of the meaning of our statements?) It will not be simple, but remember: a program of more than, say, 20 lines is probably too long, we then have not chosen the right subcomponents.

For the formulation of connection patterns we introduce the term node. Every port is a node, but the program may introduce additional (interior) nodes. For each node  $n$  we shall introduce a connection condition  $C(n)$  and a connected-to-constant condition  $CC(n)$ . We shall, furthermore, distinguish a directly driven set  $D$ , which is a subset of the set of nodes. These concepts will be used in the syntax checking. A formal definition of how they depend on the connection pattern specified will be given later. Intuitively,  $C(n)$  will be the condition on the input values under which node  $n$  is connected to an input, and  $CC(n)$  will be the condition under which it is connected to a constant. The  $C(n)$ 's will be used to enforce the no-fighting rule. The set  $D$  will comprise all nodes that are connected by a wire path to an internal input port.

The program consists of a sequence of statements. Each statement introduces a number of connections and switches between nodes, and thereby affects the  $C(n)$  and  $CC(n)$  of each node involved and the set  $D$ . Initially, i.e., prior to the first statement,  $D$  is the set of all internal input ports,  $C(n)$  is 1 for each input port and  $CC(n)$  is 1 for the two constant internal input ports. The  $C(n)$  and  $CC(n)$  are 0 for all other nodes. ("1" should be interpreted as "true" and "0" as "false.")

The program is complete if finally we have:

for every external output port  $p$  :  $p \in D \vee CC(p) = 1$   
for every internal output port  $p$  :  $C(p) = 1$

(These completeness conditions correspond to Rules 2 and 3. The observing of Rule 1 is discussed below.)

EXAMPLE 1

```
comp inverter (in?,out!):
begin in' → out = 1; in → out = 0 end
```

The above is a simple example of an RL, it does not have subRL's. The first line specifies the name of the component and its external ports. A question mark or an exclamation point indicates that the port is an input port or an output port, respectively. In the connection pattern two switches are specified, textually separated by a semicolon. The first statement expresses that the output port out is connected to the constant input port 1. The condition in front of the arrow specifies under which circumstances the switch in the connection should be on. In this case a normally-on switch whose gate is connected to the input port in (or a normally-off switch with its gate connected to in') is specified. The second statement specifies the second switch.

For the more pictorially inclined reader we observe the resemblance of the program and the following diagram.

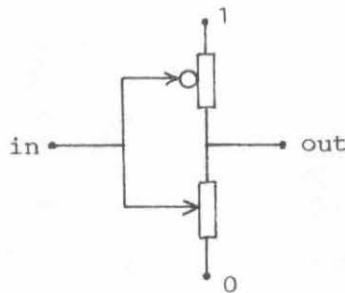


Fig. 8

Why is the program syntactically correct? In order to be able to show that the only output port out satisfies

$$\text{out} \in D \vee CC(\text{out}) = 1$$

we have to be more precise as to how a statement affects  $C(n)$ ,  $CC(n)$  and  $D$ .

In a program switches are introduced by statements

$$BE \rightarrow x = y$$

in which  $x$  and  $y$  are nodes, and  $BE$  is a boolean expression in terms of nodes, more precisely:  $BE$  is a production of the grammar

```

<boolean expression> ::= <term> { v <term> }
<term> ::= <factor> { ^ <factor> }
<factor> ::= <primary> | <primary>'
<primary> ::= <node> | (<boolean expression>)

```

Prior to the statement

$$BE \rightarrow x = y$$

we should have

for all nodes  $n$  in  $BE$  :  $C(n) = 1$ , and

$$(C(x) \wedge C(y) \wedge BE) = 0$$

The first requirement is introduced to permit the syntax checking to be done incrementally at each statement of the program. A consequence, however, is that not every order of the statements in the program is permissible. It is still an open question whether this serializability requirement is not too strong. If we succeed in designing our components under this regime it will certainly enhance both the readability and the checkability of our programs.

The second requirement guarantees the observance of the no-fighting rule. The statement does not have an effect on the set  $D$ . The effect on  $C(n)$  and  $CC(n)$  is

$$Z(x) := (Z(x) \vee (Z(y) \wedge BE))$$

$$Z(y) := (Z(y) \vee (Z(x) \wedge BE))$$

in which  $Z$  stands for  $C$  or  $CC$ .

The set  $D$  is affected only by a statement that specifies a direct connection, i.e., one that does not go through a switch. We obtain such a statement by dropping the conditional part " $BE \rightarrow$ ":

$$x = y$$

As for the effect on  $C(n)$  and  $CC(n)$  this statement is like a switch specification with "1" as its boolean expression. Prior to the statement the condition

$$(C(x) \wedge C(y)) = 0$$

should hold, and its effect is that  $Z(x)$  and  $Z(y)$  both become  $Z(x) \vee Z(y)$  ( $Z$  still standing for  $C$  or  $CC$ ). The effect on the set  $D$  is that if either node  $x$  or node  $y$  was a member of  $D$  then  $D$  is extended with the other node.

In the example of the inverter we initially have out  $\notin D$ . As the program leaves the set  $D$  unchanged we have to show that it establishes  $CC(out) = 1$ . The first statement is legitimate as we initially have  $C(in) = 1$  and

$$\begin{aligned} C(out) \wedge C(1) \wedge in' &= 0 \wedge 1 \wedge in' \\ &= 0 \end{aligned}$$

The effect is that both  $C(out)$  and  $CC(out)$  become  $in'$ . The second statement is legitimate as well:  $C(in)$  is still 1 and

$$C(\text{out}) \wedge C(0) \wedge \text{in} = \text{in}' \wedge 1 \wedge \text{in} \\ = 0$$

It establishes  $CC(\text{out}) = \text{in}' \vee \text{in}$ , which is 1. Hence, it is a complete program.

Notice that both switches in the inverter are of the type that can be implemented by one transistor. The inverter, consequently, requires only two transistors. We shall use this inverter as a sub-component in our third example.

EXAMPLE 2.

```
comp nor(a?, b?, out!):
begin a ∨ b → out = 0; a' ∧ b' → out = 1 end
```

In the first statement the boolean expression is a disjunction of two nodes. This gives rise to a diagram in which two switches are placed in parallel. The boolean expression of the second statement specifies two switches that are placed in series. The whole component requires four transistors. The following picture shows a diagram of the component.

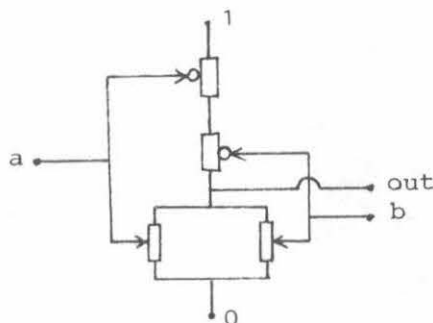


Fig. 9

A new node is introduced by mentioning it in the right-hand side (in the part to the right of the arrow) of a statement. There is no example of this in the paper.

EXAMPLE 3.

```
comp flip-flop(in?, ld?, q!, qbar!):
begin sub i1,i2: inverter;
      i2.in = i1.out;
      ld' → i1.in = i2.out; ld → i1.in = in;
      q = i2.out; qbar = i1.out
end
```

The second line of the program specifies that the component flip-flop has two subcomponents, named i1 and i2, of type inverter. As each inverter has two external ports, this declaration provides the component with four internal ports. An internal port that corresponds to the external port p of a subcomponent S is denoted as S.p. As both i1 and i2 have an external output port out, the component flip-flop has the internal input ports i1.out and i2.out. Likewise, it has the internal output ports i1.in and i2.in.

The reader is encouraged to check that the component satisfies the rules by formally deriving that all statements are legitimate and that the program establishes

$$q \in D, \text{qbar} \in D, C(i1.in) = 1, C(i2.in) = 1$$

A possible diagram of the component is

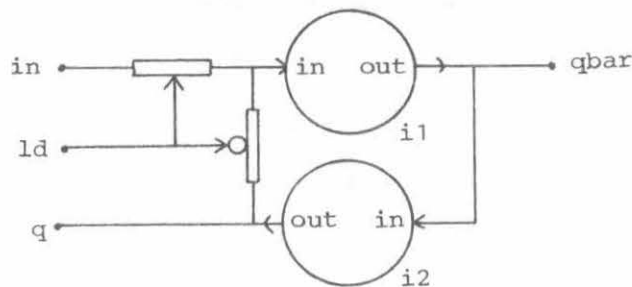


Fig. 10

## 5. BUSES

If we want to design a random access memory out of inverters, we must be able to connect their inputs and outputs via buses to the inputs and outputs of the memory. We want to connect the outputs of many subcomponents (inverters) to the same bus. Just connecting these outputs (internal inputs to the memory) to the bus would violate the no-fighting rule. We shall remedy this by putting switches in these connections.

To indicate when the memory cell has to drive the bus ("reading") and when it has to receive a value from the bus ("writing") two inputs,  $r$  and  $w$ , go into the cell:

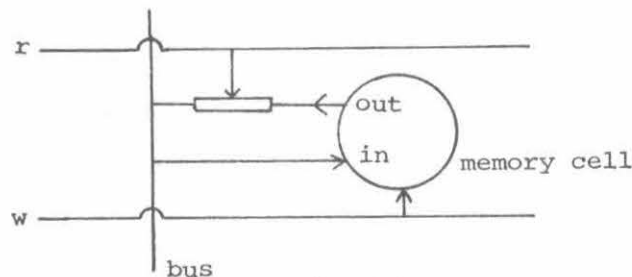


Fig. 11

We attach a number of cells to the same bus. Such a composition will only be an RL if we guarantee that, at most one of the cells can have its  $r$  equal to 1. The signals  $r$  come from another subcomponent of the memory, usually called the "decoder." The purpose of the decoder is to assure that at most one  $r$  equals 1. Given that the outputs of the decoder satisfy that requirement, we can show that the composition is again an RL. This is a new phenomenon: a condition on the values output



by a subcomponent has to be taken into account to prove that a connection pattern specifies an RL. We call such a check a semantic check.

The following program is a 1-of-2 decoder.

```

comp 1-of-2 decoder(in?, out1!, out2!):
  begin in → out1 = 1; in → out2 = 0;
        in' → out1 = 0; in' → out2 = 1
  end

```

By a syntactic check, as described in Section 4, we can show that this is a legitimate RL. In this case it is also simple to check that the output values satisfy  $(out1 \wedge out2) = 0$ , but that is a semantic check.

The moral is that we will design components that are only "conditional RL's," i.e., they are RL's under the condition that the output values of other components satisfy certain constraints. When such components are put together we will have to see to it that such semantic constraints are indeed satisfied.

## 6. A GLANCE INTO THE FUTURE OF COMPUTING

In this paper we have not addressed the dynamic behavior of components, i.e., how they react to transitions on their inputs. That is obviously the next step. By adopting proper timing and signaling conventions (cf. Chapter 7 of [2]) one should be able to address the dynamic behavior in an equally discrete fashion. The purpose of such conventions is to generate "data valid" inputs that signal that the input data are well-defined and may be inspected. Such a data valid signal may come from a clock or it may be an asynchronous acknowledge signal.

After that there are two roads we can follow. We can make a machine. That machine will accept programs and execute them. We then concentrate on the programs and if we wish to have a certain computation performed, we write a program for it. That is the traditional road.

We are led to the other, more promising, road if we observe that we are already designing programs, programs that can be compiled into transistor diagrams for CMOS. We make components out of subcomponents. Every time they will be more "powerful" or "sophisticated" than their subcomponents. We can inspect how a component is implemented by looking at its program text to see how it is composed out of subcomponents. Every component is again an implementation of a "higher level" concept. We can, e.g., introduce components that communicate other data types than just 0's and 1's. If we look at the implementation of that concept, we may notice that it is achieved by multiplexing or by the use of multiple ports. In that way the components we introduce will give us new modes of expression so that we can formulate our programs in terms of concepts that are more appropriate to our computations. After a while, we will have a mode of expression that one would customarily call a "higher level programming language."



Throughout all the levels of the hierarchy we have maintained that we program by composing components out of communicating sub-components. But by expressing a program in such a notation we have also specified an implementation for it, we have actually specified for the program a transistor diagram in CMOS. From there, the step to a complete silicon compiler is a (nontrivial) matter of generating the proper geometric representation of the transistor diagrams.

Of course, we do not have to translate all our programs into silicon to have them executed. We could also compile them into machine code, e.g., into code for a machine designed by taking the other aforementioned road. Our choice will depend on such external factors as the speed with which the computation has to be performed or the expected frequency of its use. It is also possible that we want to make a translation into machine code first in order to get some experience with the program and that we do not have it compiled into silicon until it is in a form that suits us.

#### POSTSCRIPT

Is this an article about machine design or about programming? The answer to that question is definitely "Yes!".

#### ACKNOWLEDGEMENTS

The research described in this paper was sponsored by the Defense Advanced Research Projects Agency, ARPA Order Number 3771, and monitored by the Office of Naval Research under contract number N00014-79-C-0597.

#### REFERENCES

- [1] Johannsen, Dave, "Hierarchical Power Routing." Display file 2069, Computer Science Department, California Institute of Technology, Pasadena, CA, October 1978
- [2] Mead, Carver & Lynn Conway, "Introduction to VLSI Systems." Addison-Wesley Publishing Company, Reading MA, 1980



## A STRUCTURED APPROACH TO VLSI LAYOUT DESIGN

M.S.KRISHNAN

XEROX Corporation  
LSI Development A3-74  
701 S.Aviation Blvd.  
El Segundo, CA 90245

### ABSTRACT

A new approach to the VLSI layout problem is proposed that produces a structured floor plan for an arbitrary network of interconnected processing elements. It is based on extracting a minimum spanning tree from a given representation of a computation network and using an efficient, structured layout scheme for this minimum spanning tree. Techniques to lay out trees as arrays of layout slices are presented. It is assumed that the nodes of a network are identical in their layout size and connectivity. This method is valid at any level of a VLSI design since these nodes may represent gates, cells or complex macros. An application of this approach to modified tree networks is described. Other useful applications of the method are mentioned.

## 1. INTRODUCTION

A significant portion of a VLSI chip of any reasonable complexity is consumed by the communication paths among the various macros comprising the chip. This situation is further aggravated by several factors:

- a) Decreasing feature sizes, e.g. transistors and wires, which cause communication delays to decrease non-proportionately with gate delays.
- b) Presence of random logic with the attendant interconnection that is also irregular.
- c) Lack of adequate design aids that guide a designer along a structured, hierarchical design sequence that is also streamlined to provide masks within a short time.

Traditionally, the two major phases of digital system design, namely logic design and physical, or more appropriately geometric design, have been treated as separate, sequential operations. This methodology was generally adequate before the LSI revolution. With the enormous computing power and hence complexity present in a network of processors in one chip, the separation of these two tasks tends to overlook the effects of one on the other. It is essential for a VLSI designer to be able to evaluate the effects of his logic design on the layout and vice versa early enough to incorporate them into his design. Yet there is a lack of adequate design aids for evaluating the potentials of alternative chip designs carried through the logic design to the floor plan of the chip.

The major goal of the work described in this paper is the development of design aids that will produce structured chip layouts that are efficient in area and are easy to generate. The problem of regular and/or area-efficient layouts for VLSI has generated considerable interest recently [7]. The Bristle Blocks approach [6] attempts to develop cells that have built-in stretching points so that neighboring cells may be made to conform to the same pitch. Leiserson [2] describes a divide-and-conquer approach to the layout problem wherein any planar graph that satisfies the conditions of the separator theorem of Tarjan and Lipton may be recursively bisected by removing edges until subgraphs realizable as rectangular layouts with the desired aspect ratio are obtained. These are then recursively connected by restoring the deleted edges. Brent and Kung [1] proposed a regular layout for a carry-lookahead addition scheme.

The present work attacks the problem through the creation of regular "layout slices" for a few commonly found computation structures -- tree, cube, hexagonal array etc. and treating a given computation network as a composition of instances of these structures. Thus the tasks of logic design and geometric design are being absorbed into one wherein the impact of one on the other can be handled systematically.

Section 2 describes some regular layout schemes for trees proposed in the literature. Section 3 presents some new layout schemes that are implementable as arrays with useful properties. These schemes are applied to modified tree networks in Section 4. Section 5 describes an algorithm to generate the floor plan for an arbitrary computation network of interconnected processors. Some potential applications of this method are listed in Section 6.

## 2. LAYOUT SCHEMES FOR TREES

A structure that has a wide range of applications from multiplexors, decoders etc. to multiprocessors is a tree network. A regular layout for the carry chain computation in an adder has been proposed [1]. It is actually a set of trees with common inputs and is illustrated in Fig. 1. Its area is  $O(n \log n)$  where  $n$  is the number of leaf nodes.

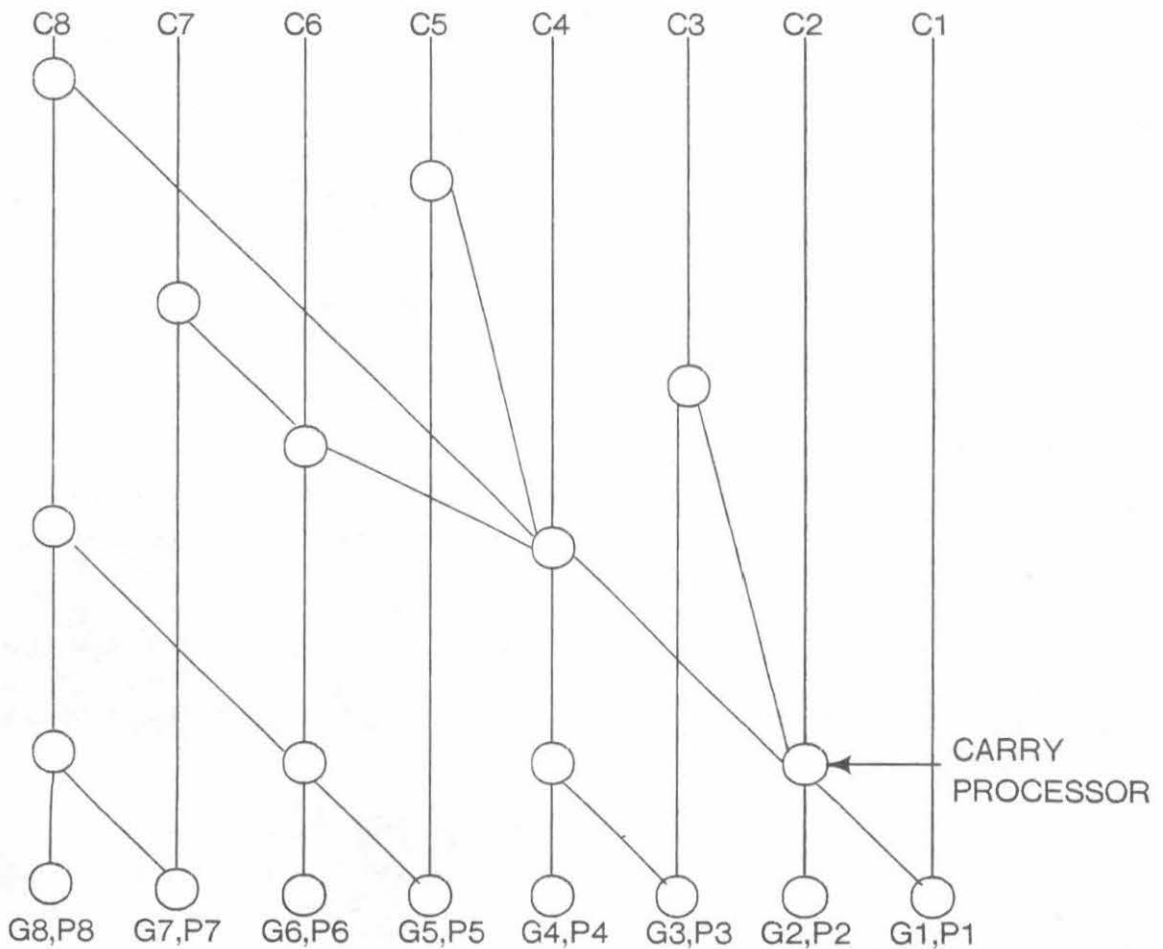


Fig. 1. An  $O(n \log n)$  layout for a carry lookahead adder tree

An H-tree layout for binary trees has been proposed [9] that is more space-efficient. The H-tree requires  $O(n)$  area and is shown in Fig. 2.

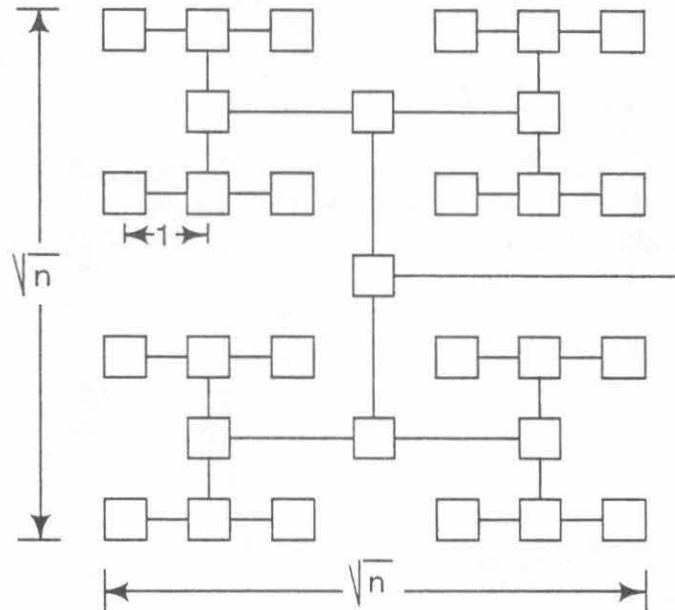


Fig. 2. The H-Tree Layout for a Complete Binary Tree

Several observations can be made on these layout schemes :

- The tree layout scheme of Fig. 1 has all the leaf nodes at one end and the output nodes at the other. Thus data travels in one direction only and a total distance proportional to  $\log n$ .
- The H-tree scheme has leaf nodes spread throughout the interior as well as the periphery of the square area. The data flow alternates between the two directions from level to level. The total distance traversed by data from the leaf nodes to the output, using the unit shown in Fig.2, is :

$$T_d = \begin{cases} 1.5 \cdot 2^k & \text{for } n = 2^{2k} \text{ leaf nodes} \\ 2^{k+1} & \text{for } n = 2^{2k+1} \text{ leaf nodes} \end{cases}$$

This delay is proportional to  $\sqrt{n}$ .

c) Both schemes grow in both directions as the tree expands. In both, the number of nodes in either direction is not constant and varies across the layout. Therefore neither scheme is suitable for realization of a tree as a one-dimensional array.

### 3. LAYOUT SLICE SCHEMES FOR TREES

We develop, in this paper, a structured layout scheme as a one-dimensional array of "layout slices". The new layout technique places the leaf nodes along the edges for ease of routing, minimizes the data propagation time so that the latency time of the tree as a segment in a pipeline is minimized. The ease of access to the leaf nodes is critical in applications where not only the root but also the leaf nodes of the tree communicate with other macros on the chip.

#### 3.1 BINARY TREES

An algorithm to generate the layout for a complete binary tree is given below. The numbering notation used in this paper for levels of nodes is shown in Fig. 3.

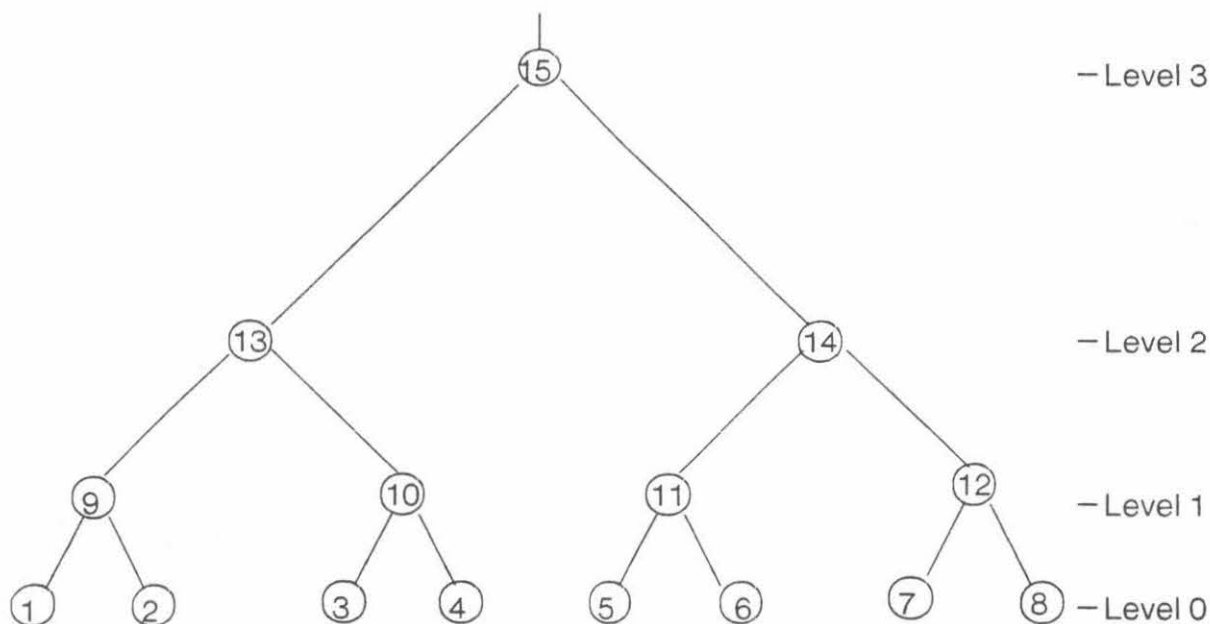


Fig. 3. A Logical Binary Tree with Eight Leaf Nodes

**ALGORITHM 1** (Algorithm for the layout of a binary tree) :

Let the binary tree have  $n = 2^k$  leaf nodes for some integer  $k$ . The two main tasks in the layout process are placement of the nodes and interconnections among them.

**1. PLACEMENT**

- a) Traverse the tree in order.
- b) Group the nodes in the traversal into pairs.
- c) Assign every pair obtained above to a layout slice.

**2. INTERCONNECTION**

- a) The connections to the leaf nodes (level 0) are straightforward since they receive external inputs.
- b) For nodes at higher levels in the tree, the level number of a node in a given slice can be determined in a simple manner. For nodes at level 1, the inputs are from within the same slice and the slice immediately to its right. For nodes at level  $i$ ,  $i > 1$ , the inputs are from the slices  $2^{i-2}$  positions to the left and  $2^{i-2}$  positions to the right.  $\square$

The algorithm is illustrated for a tree with  $n = 8$  in Fig. 4. Step 1 of the algorithm yields (1,9) (2,13) (3,10) (4,15) (5,11) (6,14) (7,12) (8). Step 2 can be observed from Fig.4 which shows the realization of the tree.

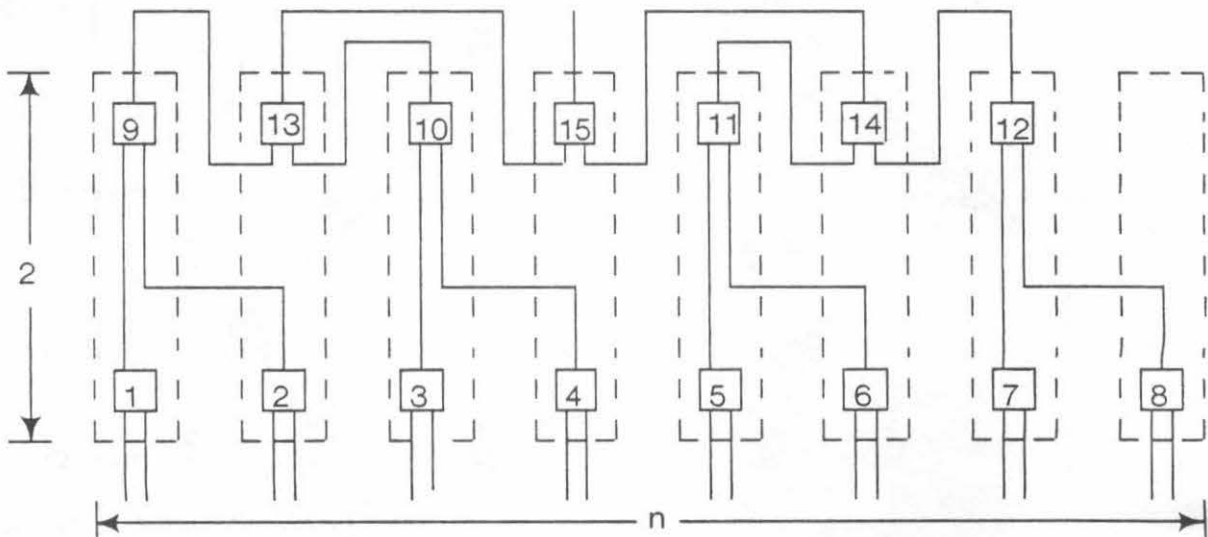


Fig. 4. Realization of a tree as an array of "layout slices"



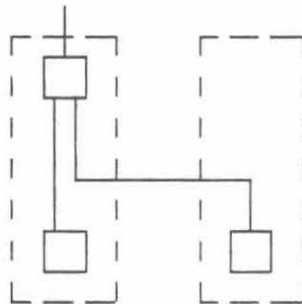
The following lemma determines the number of slices required for such a layout and the bounds on the degree and eccentricity of a slice.

**LEMMA 1 :**

A complete binary tree with  $n$  leaf nodes can be realized in area  $O(n)$  as an array of  $n$  layout slices where each slice contains exactly two nodes with a) at most four wires connected to it and b) at most  $\log n + 2$  wires passing around it without connection, where a wire is an interconnect between two nodes in the tree.

**Proof:**

We prove the first part of the lemma by induction on the number of leaf nodes. For a complete tree,  $n = 2^k$  for some integer  $k$ . For  $n = 2$ , we need two slices as shown below:



Consider a tree with  $2^m$  leaf nodes.

$$\begin{aligned} \text{The total number of nodes in the tree} &= 2^m + 2^{m-1} + \dots + 2^1 + 2^0 \\ &= 2^{m+1} \cdot 1 \\ &= 2(2^m) \cdot 1 \\ &= 2(n) \cdot 1 \end{aligned}$$

This shows that the nodes of the tree can be assigned to the  $2^m$  slices, two nodes to a slice, such that exactly one half of one of the slices is unused.

We now apply induction to a tree with  $2^{m+1}$  leaf nodes. Let

$$T_{m+1} = \begin{array}{c} R \\ \swarrow \quad \searrow \\ T_L \quad T_R \end{array}$$

be a complete tree with  $2^{m+1}$  leaves where  $R$  is the root node and  $T_L$  and  $T_R$  are the left and right subtrees of  $T_{m+1}$  with  $2^m$  nodes each. By the induction

hypothesis, both  $T_L$  and  $T_R$  have structured layouts with  $2^m$  slices each. But we have shown above that the layout for  $T_L$  has a slice, say  $S$ , that has an unused node. Let  $R$  be assigned to this unused slot and connected to the roots of  $T_L$  and  $T_R$ . This results in a layout for  $T_{m+1}$  as an array of  $2^{m+1}$  slices with each slice containing two nodes. An interesting property of the placement strategy is that every slice has exactly one leaf node and one node from a higher level. This follows from the in-order traversal of the tree. Thus the area of the layout, using any choice of units is  $O(n)$ . The maximum propagation length from the leaf nodes to the root is  $n/2$ . The maximum number of wires across any vertical cross section of the layout is  $\log n$ .

b) Let the levels of the tree be numbered with the leaf nodes at level 0 and let  $l(P)$  denote the level of node  $P$ . As mentioned above, one of the two nodes in a slice is from some level  $> 0$ . Let  $P$  be such a node. There can be no wires passing around the slice containing  $P$  that connect to a node at level  $l(P)$  or less since, by construction, none of the left sons of node  $P$  are to the right of  $P$ . The maximum number of wires passing around a slice occurs for the case  $l(P) = 1$  for which there are  $\log n - 1$  levels above it. However, the root node is not connected to any other slice and hence there are at most  $\log n - 2$  wires passing around a slice. The number of wires connected to a slice is maximum when  $l(P) \neq 1$  and can be seen to be four.  $\square$

The number of wires passing around a slice was treated specifically above to bound the width of the routing channels although these wires may be run through the cells. Only one half of one slice is unused. Note in Fig. 4 that there are at most  $\log 8 - 1$  i.e. 2 wires passing around a slice. The output from a node at level  $i$  passes around  $2^{(i-1)} - 1$  slices using this arrangement of nodes.

### 3.2 TREES WITH LARGER FANOUT

The above scheme for binary trees can be generalized to any  $k$ -ary tree as stated in the following lemma.

#### LEMMA 2 :

Any  $k$ -ary tree where each node has  $k$  inputs and one output, with  $n$  leaf nodes can be realized in area  $O(n)$  as an array of  $n$  layout slices where each slice contains at most two nodes with a) at most  $k + 2$  wires connected to it and b) at most  $\log n - 2$  wires passing around it without connection.

**Proof:**

The construction of a  $k$ -ary tree layout is similar to that of a binary tree. At each level  $i$ , a node is placed in the same slice as its  $\lceil k/2 \rceil$ th input for  $i = 1$  and in the slice  $k^{i-2}$  positions to the right of its  $\lceil k/2 \rceil$ th input for  $i > 1$  so as to place at most two nodes in every slice. The properties of this layout follow from arguments similar to those for Lemma 1.  $\square$

The placement criterion stated above is illustrated for a ternary tree with  $n = 9$  in Fig. 5. Note that this criterion is also true for a binary tree.

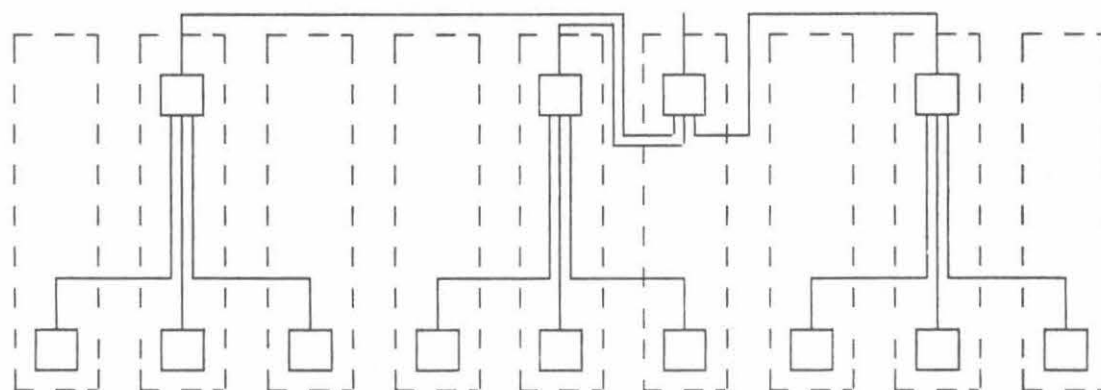


Fig. 5. Ternary tree layout as array of slices with two nodes each

The above layout schemes have two noteworthy properties:

a) There are two distinct types of slices, characterized by the I/O connections of their nodes. These can be represented for the binary tree as



Although  $S_A$  can be obtained from  $S_B$ , we choose to distinguish them in the following. These two slice types alternate in the array.

b) The number of nodes in a slice is not restricted to 2 as described in Lemmas 1 and 2, but can be any power of 2.

Some useful implications of these properties are stated in the following lemma.

**LEMMA 3 :**

A tree network can be realized as an array of two distinct types of slices, denoted by  $S_A$  and  $S_B$ , where  $S_A$  consists of a left subtree and its parent node, say  $F$ , and  $S_B$  consists of the right subtree of  $F$  and an ancestor of  $F$ . The array realization is an alternating sequence of these two slice types. There are  $\log n + 1$  different realizations of the tree, characterized by the number of nodes in a slice, where  $n$  is the number of leaf nodes.

**Proof:**

The basic unit in a binary tree is a node along with its two children. Out of the three ways of assigning these three nodes to two adjacent slices, the ones that yield the minimum inter-slice communication assign the parent node and one of its subtrees to be in the same slice as its parent. We have arbitrarily chosen the left subtree to be in the same slice as its parent. The sequence  $S_A, S_B$  by this definition, expands the left subtree contained in  $S_A$  to the next higher level while the sequence  $S_B, S_A$  inserts the right subtree of the parent node contained in  $S_B$ . The result follows from an inductive argument on the sequence of slice types. With  $n = 2^m$  leaf nodes, there are  $2^{m+1} - 1$  nodes in the tree. A slice may contain  $2^i$  nodes,  $1 \leq i \leq m+1$ . Each of these produces a distinct realization and hence there are  $m+1$ , or equivalently,  $\log n + 1$  different realizations of the tree.

It should be pointed out that the layout slice arrangement is amenable to a gate array in which a "gate" may be a layout slice and the number of interconnecting channels may be bounded as above.

### 3.3 AN ALTERNATIVE REALIZATION OF A BINARY TREE

Another layout slice scheme for complete binary trees that uses a single slice type is briefly described below. Each slice contains a basic unit of the tree.

**LEMMA 4 :**

A complete binary tree with  $n$  leaf nodes can be realized as an array of  $\lceil (2n-1)/3 \rceil$  identical layout slices where each slice contains a parent node and its two children, with a) at most five wires connected to a slice and b) at most  $\log n - 2$  wires passing around it without connection.

**Proof:**

- a) As shown in Lemma 1, the total number of nodes in a tree with  $n$  leaf nodes is  $2n-1$ . Assigning a parent node and its two children to a slice results in  $\lceil 2n-1/3 \rceil$  slices and at most five wires connected to a slice.
- b) The bound on the number of wires passing around a slice can be proved by induction on the number of leaf nodes.  $\square$

This scheme also has the leaf nodes at the edges of the layout and is illustrated in Fig. 6 for  $n = 4$  and  $n = 8$ . It can be observed that the slices are fully used when  $m$  is odd and there are exactly two unused slots when  $m$  is even, where  $n = 2^m$ .

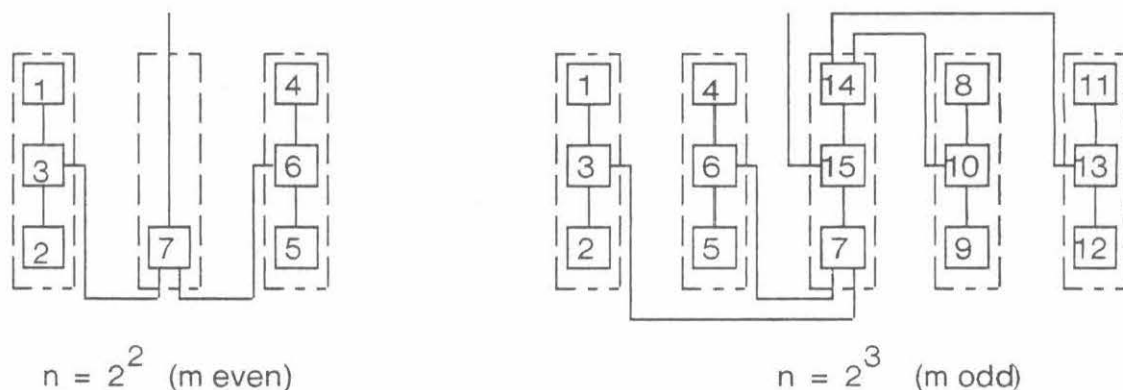


Fig. 6. An alternative array realization of the tree with one slice type.

#### 4. MODIFIED TREE NETWORKS

Let us consider the applicability of the array-realizable layout schemes to modified tree networks. Such a network of considerable interest is a carry-save adder that adds a set of  $n$ -bit numbers using standard full/half adder cells with the carry propagation deferred to the very last stage. The number of levels in this tree is determined by the type of basic adder cells used in reducing the given  $h$   $n$ -bit numbers to two rows of bits before performing a carry propagation. For instance, using full/half adder cells for each node of the network one would require approximately  $\log_s h$  levels in the tree excluding carry propagation, where  $s \approx 1.5$  [10]. It suffices to say that the number of levels in the tree for a given type of adder cell can always be bounded.

An assignment of full/adder cells to add six 3-bit numbers is shown in Fig. 7. The

horizontal lines separate the successive levels in the tree. The numbers within circles are the unit numbers of the adder cells and the numbers within the adder cells (boxes) represent the outputs of other adder cells from previous levels. There are four levels in the tree and the carry propagation is done at the fourth level.

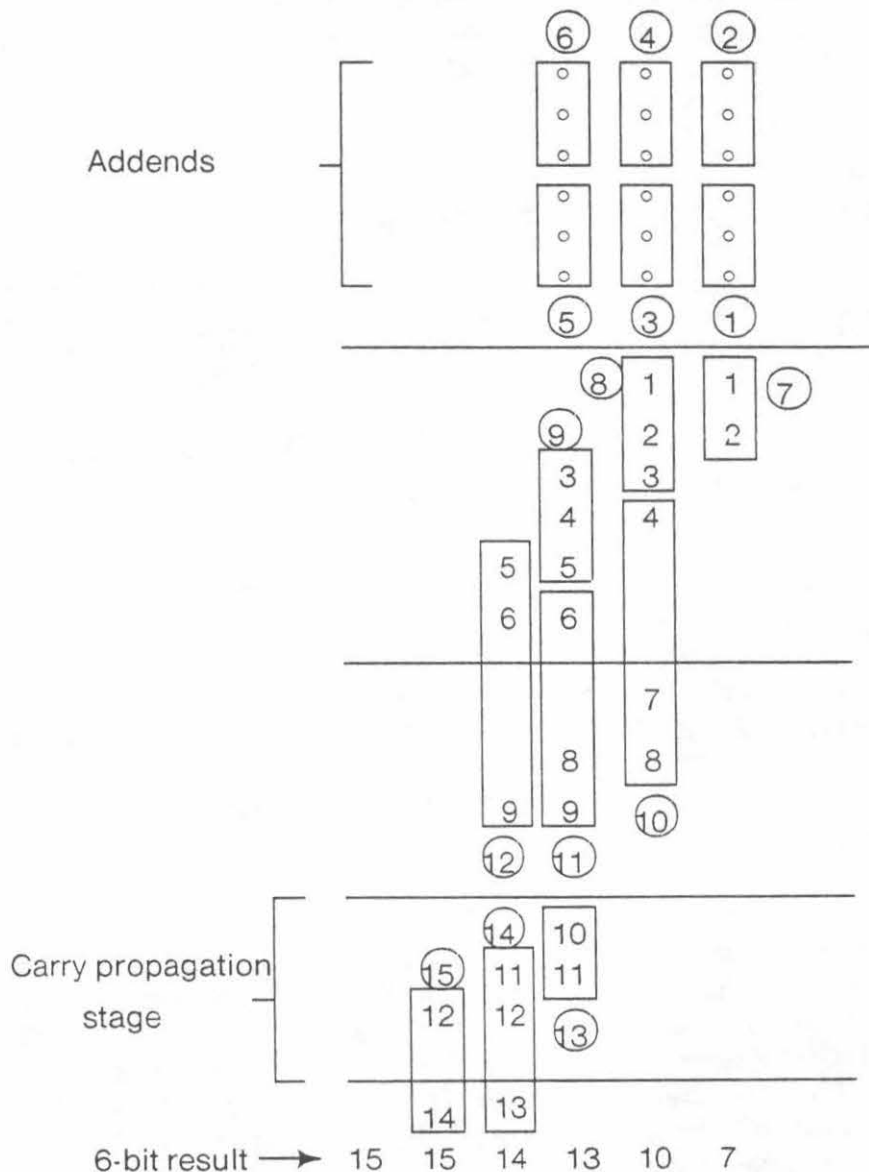


Fig. 7. Carry-save addition of six 3-bit numbers using full/half adders

The tree network for this adder scheme is shown in Fig. 8. Let us assign the nodes in the tree to layout slices as follows:

Define a slice for each leaf node. Assign a non-leaf node  $P$  to the same slice as its middle input if  $P$  is a full adder and to the same slice as its right input if  $P$  is a half adder. Note that there is at most one node from any given level in a slice. The placement of the carry propagation stage cells is explained below. The effect of this assignment strategy is that in any slice there is at most one node from any given level. A more rigorous method for obtaining the layout slices for general networks will be described in Section 5. We can now state the following for such an adder tree implementation using layout slices:

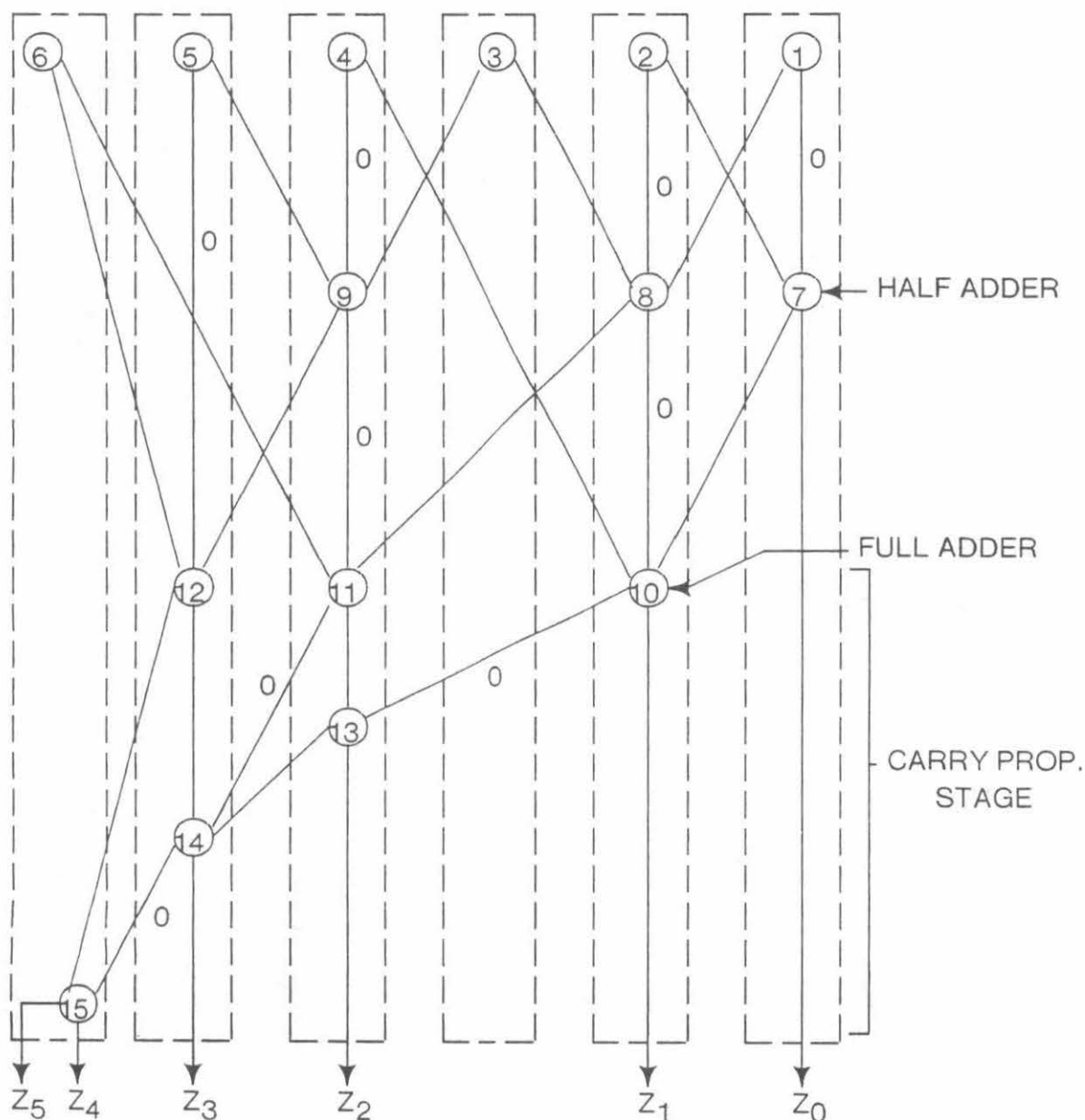


Fig. 8. Tree Network for Carry-save Addition using Full/Half Adders



**LEMMA 5 :**

An adder tree network that adds  $h$   $n$ -bit words using carry-save addition with full/half adder cells can be realized in area  $O(n \log_s h)$  as an array of  $n + \lceil h/2 \rceil$  slices with each slice containing at most  $\lceil \log_s h \rceil + 1$  nodes and at most  $5 \lceil \log_s h \rceil + 4$  wires connected to it, where  $s \simeq 1.5$ .

**Proof:**

As discussed above, the upper bound on the number of levels in the tree using full adders is  $\log_s h$ . For the carry propagation, we need at most  $(n-1) + \lceil h/2 \rceil$  more levels where the term  $\lceil h/2 \rceil$  accounts for the fact that the sum of a pair of  $n$ -bit numbers will need an additional bit for overflow and so the sum of the  $h$   $n$ -bit numbers may have up to  $n + \lceil h/2 \rceil$  bits. For simplicity, we allow these additional  $\lceil h/2 \rceil$  bits of the sum to have separate slices for the carry propagation stage. Thus we need at most  $\lceil \log_s h \rceil + 1$  nodes in each slice including the carry propagation stage. Each of the nodes in a slice has two or three inputs and two outputs. However, from the assignment of the nodes to slices, we are guaranteed that at least one node in every slice has at least one input from within the same slice. Thus there are at most  $5(\lceil \log_s h \rceil + 1) - 1$ , i.e.  $5\lceil \log_s h \rceil + 4$  wires connected to any slice. The area of the layout is therefore  $O(n \log_s h)$ .  $\square$

The dotted lines in Fig. 8 indicate the slices used in the realization of the tree.

**5. A FLOOR PLAN GENERATION APPROACH**

In this section an approach to develop the floor plan of an arbitrary computation network of interconnected processors is outlined. The task of generating a floor plan is to lay out the individual nodes and the interconnections among them in a rectangular area satisfying the specified design constraints like line length, width and number of crossovers. The availability of alternative layout schemes is bound to suggest alternative logic realizations at any level of a hierarchical design. The layout schemes described above are not limited to any particular logic level, e.g. transistors, gates etc. Thus various multiprocessor architectures as well as different multiplication schemes may be tried with the same abstraction. The elementary nodes themselves may in turn be laid out in detail at a lower level to the desired degree of optimization.



## 5.1 DESCRIPTION OF THE METHOD

The approach consists of the following steps:

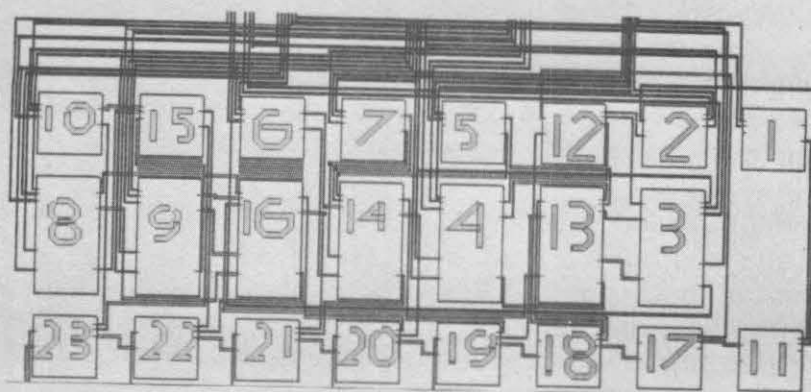
- a) Obtain a weighted network of processing elements with appropriate weights assigned to the edges. These weights may represent the required degree of proximity of the two nodes connected by that edge.
- b) Extract a minimum spanning tree for the network i.e. a tree that spans all nodes of the network and has the minimum total weight for any tree. The designer can force critical interconnections into this tree by assigning appropriate weights to these edges in the original network. The resulting minimum spanning tree will contain the part of the original network that is critical in terms of topology constraints.
- c) Lay out the minimum spanning tree obtained using the regular tree layout schemes described above.
- d) Realize the original network by restoring the remaining edges.

The implementation of the above method is illustrated in Fig. 8. The leaf nodes are at one end and each leaf node defines a slice. The middle input to a full adder and the right input to a half adder have been assigned weight 0. The effect of this criterion on the assignment of nodes to slices is evident although the actual strategy for the assignment of weights to the edges is not relevant to the proposed floor plan method.

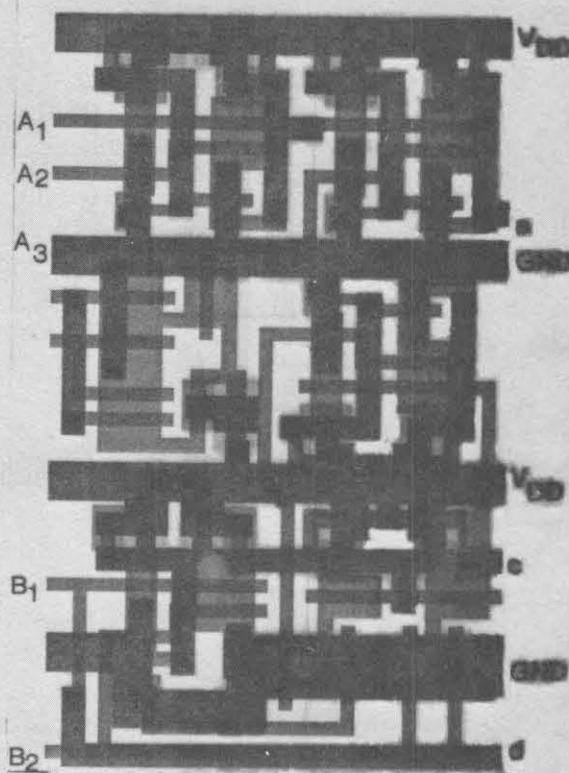
The layout of a bigger version of a carry-save adder macro generated using the above method is shown in the accompanying plate which shows the overall floor plan of the macro and the internal layout of an adder cell. There are five 6-bit operands and the sum is truncated to 6 bits. The leaf nodes are in the top row and the output nodes in the bottom row. Note that the middle row consists of one type of adder cell while the top and bottom rows contain a smaller adder cell although the above method was developed primarily for identical cells. The inputs and outputs of a cell are on opposite faces of a cell. A cell layout or its mirror image may be used in a slice.

## 5.2 INCOMPLETE TREES

Although the layout schemes described above have assumed complete or nearly complete trees, incomplete or unbalanced trees may still bring useful layout structure to the original network. Fig. 9 illustrates this with incomplete trees laid



OVERALL FLOOR PLAN



LAYOUT OF ADDER CELL  
LAYOUT OF ADDER MACRO

out as arrays. The interconnections among nodes are weighted, with 0 indicating a closer required proximity than 1. External inputs are shown unweighted and undirected. Note that in (b), node 3 is not a conventional leaf node.

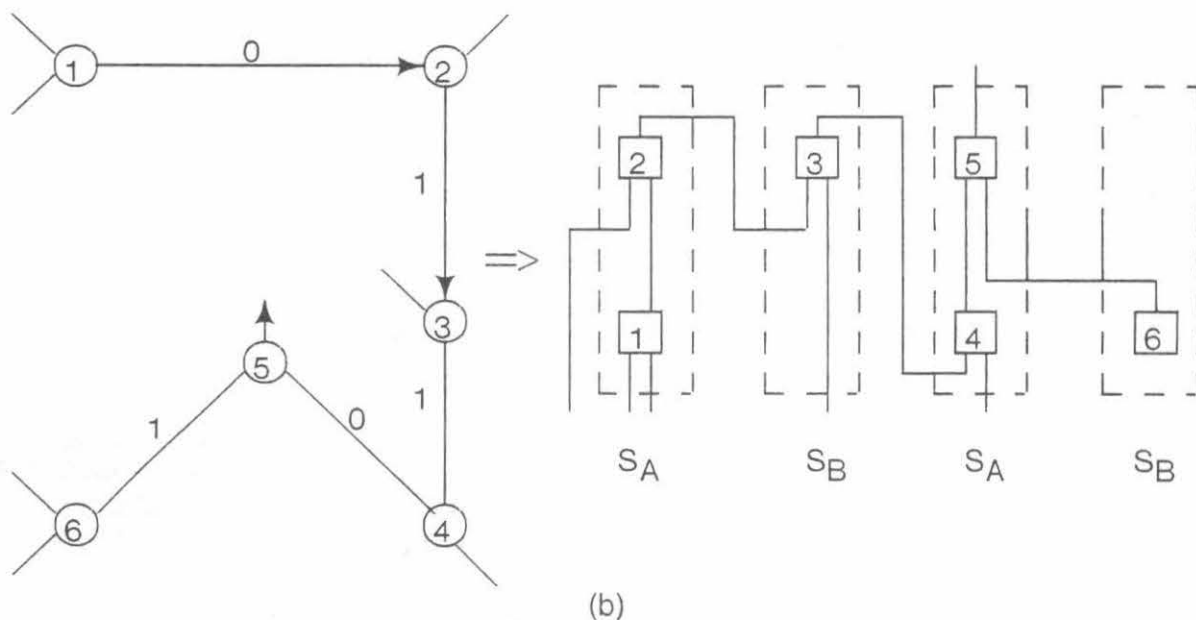
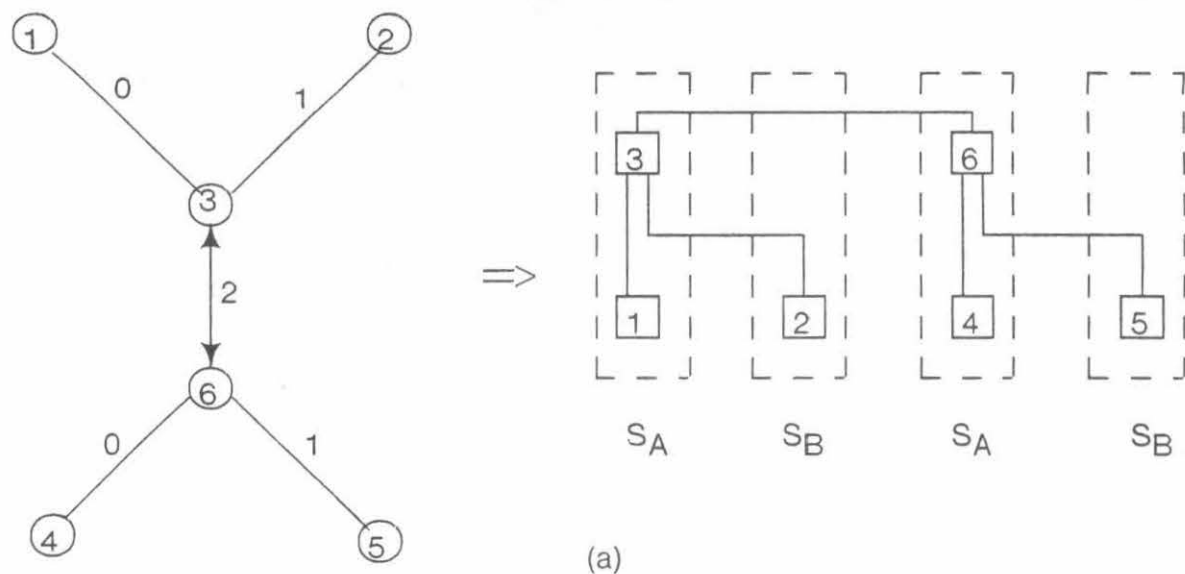


Fig. 9. Examples of layouts for incomplete trees

### 5.3 PROPERTIES OF THE PROPOSED FLOOR PLAN APPROACH

- 1) It is more general than the divide-and-conquer method underlying Leiserson's scheme in that any network, not necessarily planar, can be handled as a minimum spanning tree problem.
- 2) It yields a regular, array-realizable layout with known bounds for the number of processors in a slice and the number of wires between slices which provide for uniform spacing for routing purposes.
- 3) There are efficient, polynomial-time algorithms to extract minimum spanning trees [12]. More importantly, alternative minimum spanning trees can be easily obtained using cyclic interchange methods [11] making it possible to systematically generate alternative floor plans. A useful implication of this is that the design hierarchy may be reevaluated in the light of the floor plans generated, resulting in a modified computation network. Thus the processes of logic design and physical design can be integrated to simplify the time-consuming and often error-prone task of a detailed layout for VLSI chips.
- 4) An apparent disadvantage of this approach is that for an unbalanced tree the slices are not utilized efficiently. However, as mentioned in the previous section, regularity in layout can still be imposed on unbalanced trees. Also, at the mask generation stage, the unused portion of a slice may be eliminated so that the unused part of a slice does not consume any power.

### 6. CONCLUSIONS

Some layout schemes for tree networks and a possible solution to the floor plan generation problem using such schemes were proposed above. Possible directions for pursuing this approach are mentioned in this section:

- 1) Efficient layout slices for other commonly used structures e.g. cube, hexagonal array etc. may be developed such that different types of layout slices are compatible in terms of number of wires and/or number of processing elements in a slice. The goal here is similar to that of the Bristle Blocks project [6]. Thus a given network may be decomposed as a set of these structures which can then be laid out individually using efficient layout slices for each of these structures and interconnected. This compares favorably with the arbitrary division approach used in the divide-and-conquer method. For instance, the slice concept applied to a cube network is demonstrated in Fig. 10.

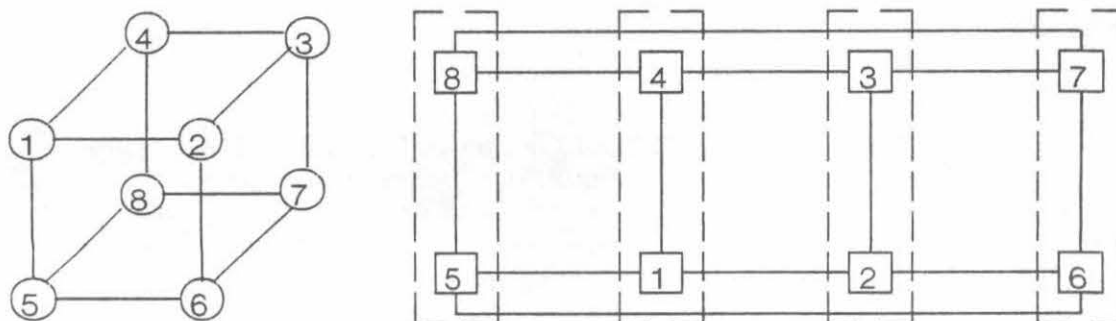


Fig. 10. Cube Interconnection realized as a regular array of slices

Note that each slice for the cube also contains two nodes similar to a tree slice and there are two slice types.

2) There are no efficient methods to extract minimum spanning trees with special constraints such as a bound on the degree of a node etc. This problem can be viewed differently: Are there useful computation networks that are modifications of a balanced tree and are realizable as arrays of slices within the bounds discussed above? The adder trees of Section 4 are examples of such networks.

3) A natural extension of the layout schemes described above would be to tree structures where the nodes are not identical in their sizes or connectivities. The selection of a basic slice or slice types would be critical to an array realization.

4) Since the processing elements within a slice may be connected internally, specific optimizations both in layout and in logic design are possible. For example, for a layout slice where a leaf node communicates its carry signal to the node within the slice, e.g. slice  $S_A$  above, it is possible to use a complemented carry signal thereby eliminating two inverters and saving their area and power. However, such optimization is meaningful only in situations where it does not cause a proliferation of layout slice types. This may be treated as a problem of characterizing the interconnection pattern among the slices.

## ACKNOWLEDGEMENT

The author is thankful to the Xerox Corporation for their support during this research.

## REFERENCES

- 1) Brent, R.P. and Kung, H.T., "A Regular Layout for Parallel Adders", Tech. Report, CMU-CS-79-131, Dept. of Computer Science, Carnegie Mellon University, June 1979.
- 2) Leiserson, C.E., "Area-Efficient Layouts for VLSI", Tech. Report, Dept. of Computer Science, Carnegie-Mellon University, August 1979.
- 3) Bentley, J.E. "Multidimensional Divide-and-Conquer", Comm. of the ACM, Vol. 23, No.4, April 1980, pp 214-229.
- 4) Rowson, J.A. "Understanding Hierarchical Design", Ph.D thesis, Dept. of Computer Science, Caltech, April 1980.
- 5) Browning, S.A., "A Tree Machine", Lambda, Second Quarter, 1980, pp 32-36.
- 6) Johannsen, D., "Bristle Blocks: A Silicon Compiler", Caltech Conf. on VLSI, Jan 1979, pp 303-310.
- 7) Marshall, M., "VLSI pushes super-CAD techniques", Electronics, July 31, 1980, pp 73-80.
- 8) Mead, C.A. and Conway, L.A., "Introduction to VLSI Systems", Addison-Wesley, Mass. 1980.
- 9) Rem, M., "Mathematical Aspects of VLSI Design", Caltech Conf. on VLSI, Jan 1979, pp 55-63.
- 10) Stenzel, W.J. et al, "A Compact High-Speed Multiplication Scheme", IEEE TC, Vol. C-26, No. 10, Oct 1977, pp 948-957.
- 11) Deo, N., "Graph Theory with Applications to Engineering and Computer Science", Prentice-Hall, Englewood Cliffs, N.J., 1974.
- 12) Cheriton, D. and Tarjan, R.E., "Finding Minimum Spanning Trees", SIAM J. of Computing, Vol. 5, No. 4, Dec 1976, pp 724-742.

## MINIMUM PROPAGATION DELAYS IN VLSI

Carver Mead  
Professor of Computer Science, Electrical Engineering and Applied Physics  
California Institute of Technology  
and  
Martin Rem  
Eindhoven University of Technology and California Institute of Technology

### 1. INTRODUCTION

With feature sizes decreasing and chip area increasing it becomes more and more time consuming to transport signals over long distances across the chip [5]. Designers are already introducing more levels of metal connections, using wider and thicker paths for longer distances. Another recent development is the introduction of an additional level of connections between the chip and the pc-board, multilayer ceramic chip carriers. The trend is undoubtedly towards even more connecting levels.

In this paper we demonstrate that it is possible to achieve propagation delays that are logarithmic in the lengths of the wires, provided the connection pattern is designed to meet rather strong constraints. These constraints are, in effect, satisfied only by connection patterns that exhibit a hierarchical structure. We also show that, even at the ultimate physical limits of the technology, the propagation for reasonably sized VLSI chips is dominated by these considerations, rather than by the speed of light.

### 2. PROPAGATION DELAY

We compute the time it takes a minimum sized transistor to drive a wire of length  $l$  with width and thickness  $s$ . We assume the wire to have a distance  $s$  to its neighboring wires and layers. Let  $s_0$  be the minimal width of a wire on the chip, so that a minimal transistor has area  $s_0^2$ .

The following equation is an excellent approximation to the total time  $T$  required to drive the wire.

$$T \approx (R_t + R_w)C_w \quad (1)$$



$R_t$  is the resistance of the minimal transistor,  $R_w$  the resistance of the wire and  $C_w$  its capacitance. The resistance of a wire is proportional to its length and inversely proportional to its cross section:

$$R_w = \rho \frac{\ell}{s^2} \quad (2)$$

The capacitance of a wire is inversely proportional to the distance of its neighboring wires and layers, and it is proportional to the area of the side facing that neighboring wire or layer:

$$C_w = \epsilon \frac{s\ell}{s} = \epsilon \ell \quad (3)$$

We notice that the product of  $R_w$  and  $C_w$  is already quadratic in  $\ell$ . Thus the time it takes to drive a wire is at least quadratic in the wire length. However, things are not as bad as they look:  $R_t$ , the resistance of a minimal transistor, is the dominant term in (1). We can decrease that term by fitting a larger driver to the wire. But that driver must then in its turn be charged by the minimal transistor and it seems that we have hardly gained anything. That, however, is not true, for we can use a sequence of drivers instead of just one. The first one is the minimal transistor, the next one is bigger by a factor  $\alpha$ . It drives another driver that is again bigger by a factor  $\alpha$ , etc., until we finally reach a driver that is large enough to drive the whole wire in a sufficiently short time.

There exists a simple rule to determine the time required to have a driver charge another driver [2]. Let  $\tau$  be the time it takes a minimal transistor to charge the gate of another minimal transistor. The rule is then that the time required to have a driver with capacitance  $C_1$  drive another driver with capacitance  $C_2$  ( $C_2 > C_1$ ) is

$$\tau \frac{C_2}{C_1} \quad (4)$$

Let  $C_t$  be the capacitance of a minimal transistor. We have it drive a driver with capacitance  $\alpha C_t$ , this second one drives a driver with capacitance  $\alpha^2 C_t$ , etc., until the last driver has a gate capacitance of about  $C_w/\alpha$ . The number of drivers (including the initial transistor) required is

$$\log_{\alpha} \frac{C_w}{C_t} \quad (5)$$



The capacitance  $C_t$  of a minimal transistor is equal to  $(\epsilon s_0^2)/d$ , in which  $d$  is the thickness of the gate insulator. The number of drivers is then  $\log_\alpha \ell d$  and we get for the time  $T_d$  spent in driving a zero resistance wire through the sequence of drivers:

$$T_d = \alpha \tau \log_\alpha \frac{\ell d}{s_0^2} \quad (6)$$

We may replace formula (1) by

$$T = T_d + R_w C_w \quad (7)$$

From (2), (3), (6), and (7) we conclude

$$T = \alpha \tau \log_\alpha \frac{\ell d}{s_0^2} + \rho \epsilon \frac{\ell^2}{s^2} \quad (8)$$

We now have a formula for the propagation delay with both a logarithmic and quadratic term. One can see why a longer wire requires a larger  $s$ : that decreases the quadratic term. Actually, we wish to restrict the lengths of wires to values of  $\ell$  that are sufficiently small to assure that the quadratic term does not dominate. We restrict ourselves to values of  $\ell$  for which the quadratic term grows at a slower rate than the logarithmic one. Therefore, we determine the value of  $\ell$  for which the derivatives with respect to  $\ell$  of the two terms are equal:

$$\frac{d}{d\ell} \alpha \tau \log_\alpha \frac{\ell d}{s_0^2} = \frac{\alpha \tau}{\ell \ln \alpha} \quad (9)$$

$$\frac{d}{d\ell} \rho \epsilon \frac{\ell^2}{s^2} = \frac{2\rho \epsilon \ell}{s^2} \quad (10)$$

If a signal has to go distance  $\ell$  we choose a path with width and thickness  $s$  for which (9) and (10) are equal:

$$s = \ell \sqrt{\frac{2\rho \epsilon \ln \alpha}{\alpha \tau}} \quad (11)$$

Substitution of (11) in (8) yields

$$T = \tau \frac{\alpha}{\ln \alpha} \left( \ln \frac{\ell d}{s_0^2} + \frac{1}{2} \right) \quad (12)$$

Or, approximately,

$$T = \tau \alpha \log_{\alpha} \frac{\ell d}{s_0} \quad (13)$$

We have assumed that the values of  $s$  could be chosen from a continuous range. Although this is a good conceptualization of the increasing number of different connection layers, in practice we will have to choose  $s$  from a discrete set. The connecting wires will be placed at different levels. The widths of the paths at the next level will be some factor  $\beta$  times the widths at the preceding level. Given a distance  $\ell$  the signal has to travel, formula (11) gives us the ideal  $s$  and we choose a level at which the widths of the wires are closest to  $s$ . This leads to an interesting observation, the "magnifying glass phenomenon:" not only will the widths of the wires at any given level be the same but their lengths will also be about equal. The patterns at different levels are similar, at the next level the features are just magnified by a factor  $\beta$ .

## 2.1 Velocity of Light

Asymptotically, no signal can travel faster than the velocity of light. We must ask under what conditions the above considerations will set a limit which is more stringent, i.e., when the velocity of light limit is not attainable. In (13) we can substitute  $\tau = s_0/v$  where  $v$  is the limiting velocity of electrons in the channel (a few  $10^6$  cm/sec in silicon)

$$T = \frac{\alpha s_0}{v} \log_{\alpha} \frac{\ell d}{s_0} \quad (14)$$

The maximum "velocity" with which signals can propagate is given by  $1/(dT/d\ell)$

$$\frac{dT}{d\ell} = \frac{\alpha s_0}{v \ell \ln \alpha} \quad (15)$$

The domain of validity of the above results is "velocity"  $< c$ :

$$\ell < \frac{c \alpha s_0}{v \ln \alpha} \quad (16)$$

For typical technology today,  $s_0 = 4$  microns,  $\alpha/\ln \alpha$  about 6 and  $\ell$  should be less than about a foot. Hence the velocity of light cannot be reached using the best MOS technology in the most optimal way within a typical small card bay, but will

be important at larger dimensions. Even for the ultimate technology ( $s_0 = 0.25$  microns), the results given above will dominate over speed-of-light considerations for chips up to about an inch across.

### 3. AREA

The arrangements outlined in the preceding section, allowing us to treat propagation delays as being logarithmic, will only work if we can allot enough area at the lowest level for the drivers and at the higher levels for the wires.

A minimal transistor has area  $s_0^2$ . The next driver in the sequence requires an area  $\alpha s_0^2$ , the third one  $\alpha^2 s_0^2$ , etc. The total area  $A$  of the drivers thus becomes

$$A = s_0^2(1 + \alpha + \alpha^2 + \cdots) \text{ (log}_{\alpha} l \text{ terms)} \quad (17)$$

$$A = \frac{s_0^2(l-1)}{\alpha-1} \quad (18)$$

Or, approximately,

$$A = \frac{s_0^2 l}{\alpha-1} \quad (19)$$

Notice that we can trade area for time. By increasing  $\alpha$  the area of the drivers decreases, cf. (19), but the propagation delay increases, cf. (13).

A transistor that has to drive a wire of length  $l$  requires area  $s_0^2 l / (\alpha - 1)$  at the lowest level. This area is proportional to the length of the wire. That is fortunate: if we double both the length and the width of a chip we also double the lengths of the longest (cross chip) wires and the areas of their drivers. But the total area of the chip will quadruple and we will thus be able to double the number of wires as well.

The longer wires come on higher levels on which the wires are wider, thereby consuming more area. Each level, however, has the same area. As a result, we can accommodate the wires at the higher levels only if we do not have too many of them. Assume again that at the next level the wires are  $\beta$  times thicker, longer, and wider. Call the lowest level number 0 and let  $N_i$  be the

number of wires at level  $i$  ( $i \geq 0$ ), then we must have

$$N_i = N_0 \beta^{-2i} \quad (20)$$

The number of wires as a function of their lengths must decrease exponentially fast. This is a strong restriction. It suggests that efficient chips must have a tree-like structure. It is again a reason to design hierarchical chips [2], [4]. If a design does not meet this exponential rule the best we can do is getting the propagation delay linear in the wire length by inserting repeaters at equidistant positions along the wires. The consequences of linear wire delays are discussed in [1].

One may also see complexity computations that assume that wires have no delay. Thompson, e.g., writes in [6]:

"The propagation time can be made independent of the length of the wire, by fitting larger drivers to longer wires. Larger drivers of course occupy more area, but need not take more than 10% of the area of the wire they drive. By fudging  $\lambda$  upwards by 5%, the area of the driver is thus absorbed into the area of its wire."

We have seen that the area of the driver is indeed proportional to the wire length, but Thompson neglects the fact that charging the gate of the larger driver will also take time. Our choice of the sequences of exponentially growing drivers allowed us to do this in a time that is logarithmic in the wire length, a technique that can work only if we have very few long wires. Thompson's model also neglects that the drivers have to be at the lowest level, in polysilicon and diffusion, independent of the level of the wire.

#### ACKNOWLEDGEMENTS

The research described in this paper was sponsored by the Office of Naval Research Contract No. N00014-76-C-0367 and by the Defense Advanced Research Agency, ARPA Order number 3771, and monitored by the Office of Naval Research under Contract number N00014-79-C-0597

4. REFERENCES

- [1] Chazell, B. M. and L. M. Monier, "Towards More Realistic Models of Computation for VLSI," *these Proceedings*
- [2] Mead, Carver and Lynn Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading MA, 1980
- [3] Mead, Carver and Martin Rem, "Cost and Performance of VLSI Computing Structures," *IEEE J. Solid State Circuits* 14, No. 2, April 1979, pp. 455-462
- [4] Rem, Martin, "Mathematical Aspects of VLSI Design," *Proceedings, Caltech Conference on VLSI*, (ed. C. L. Seitz), Computer Science Department, California Institute of Technology, Pasadena CA, January 1979, pp. 55-64
- [5] Seitz, Charles L., "Self-Timed VLSI Systems," *Proceedings, Caltech Conference on VLSI*, (ed. C. L. Seitz), Computer Science Department, California Institute of Technology, Pasadena CA, January 1979, pp. 345-355
- [6] Thompson, C. D., "Area-Time Complexity for VLSI," *Proceedings, 11th Annual ACM Symposium on the Theory of Computing*, ACM Special Interest Group on Automata and Computing Theory with IEEE Computer Society Technical Committee, Atlanta GA, May 1979, pp. 81-88



## Towards More Realistic Models of Computation for VLSI

B.M. Chazelle      L.M. Monier

Department of Computer Science  
Carnegie-Mellon University  
Pittsburgh, Pennsylvania 15213

### Abstract

We propose two new models of computation for VLSI which take into consideration the physical nature of information, the properties of wires, and the geometrical structure of the circuit. Both are refinements of the Kung-Thompson model, and make the main additional assumption that the propagation time of information is at best linear in the distance. The first is the more general and applies for any planar technology. It is in a sense the *minimal physical* model. The second, more restrictive, is specially tailored for electrical technologies. Our approach is motivated by the failure of previous models to allow for realistic asymptotic analysis. For each model, we are able to show new lower bounds and trade-offs for many well-known problems.

## 1. Introduction

The importance of having general models of computation for VLSI is apparent for various reasons. Among the chief ones, we must include the need for evaluating and comparing circuit performances, showing lower bounds and trade-offs on area, time, and energy, and more generally building a complexity theory of VLSI computation.

While these models must be simple, general enough, to allow for mathematical analysis, they must also reflect reality independently of the size of the circuit. We justify the latter claim by observing that if 1980's circuits are still relatively small, the use of high-level languages for designing chips, combined with the possibility of larger integration and bigger chips, will make asymptotic analysis necessary in the near future.

Yet as circuits are pushed to their physical limits, constraints which could be ignored before become major problems and must be accounted in the models. In particular, certain physical phenomena specific to electrical technologies enforce the density of current at any point of a conductor to be bounded. We can show that this invalidates the assumptions made in previous models, whereby long wires can be driven in constant time and an  $f$ -branch fanout takes  $O(\log f)$  time [MC80, TH79].

Generally speaking, one major flaw in those previous models is to regard a circuit as a topological interconnection of nodes where transmission delays between adjacent nodes can be ignored. Instead, we propose to take into account the geometry of the circuit by assuming a propagation delay linear in the distance. We can justify this approach by considering parameters such as length and width of wires, and associating resistance and capacitance with each part of the circuit. We will define a first model which does not make further assumptions, and we will review the complexity of some well-known circuits in this model. However, observing that in NMOS technology, the power can be supplied only from the outside boundary of the circuit, we can include this requirement and define a second model, which may be more realistic for electrical planar technologies.

Also, besides presenting new models of computation for VLSI, the purpose of this paper is to present a general technique for deriving lower bounds and space-time trade-offs for many problems, e.g., addition and transitive functions.

## 2. The Models

### 2.1. The basic assumptions

Our models are for the most part refined versions of the current planar models found in the literature [TH79, BK80, VU80]. A circuit consists of nodes and wires connected in a network, and it is defined by a geometrical layout of this network. We distinguish I/O nodes where input and output values are available, the logical nodes (gates) which compute boolean functions, and the connection nodes which simply connect



wires. The circuit is laid out within a convex region with all the I/O nodes lying on its boundary. It is the case today, and will remain true because of the greater ease in connecting and packaging such chips. In addition, we make the following set of assumptions, which define our first model (MOD1).

1. Wires have width and spacing between them greater than  $\lambda$  (today  $\lambda \approx 1\mu\text{m}$ ). This requirement will always be valid for any physical device.
2. A circuit is laid out on a finite number of layers, and wires crossing through different layers are allowed. Thus there is at most a constant number of cross-overs at any point.
3. The density of current at any point of a wire is bounded by a maximum value  $\delta_{\text{max}}$ , which is equivalent to saying that the power dissipated per unit volume is also bounded. The major consequence of this assumption is to make propagation delays at least linear in the distance.
4. To switch a gate requires a minimum energy dissipated as heat [MC80,Ch.9]; this energy must be supplied to the gate by a source other than the input signals.

To take into account the limitations in driving power enforced by NMOS and to a lesser extent CMOS technology, we introduce a second model (MOD2), which in addition to MOD1, includes the following assumptions.

1. All the energy supplied to the circuit comes from outside the circuit, and its transmission is performed only through wires. From 3, it follows that the maximum power provided to the circuit is at most proportional to the perimeter of the circuit.
2. Storing a bit of information requires a minimum energy per unit of time.

Note that since this model is more restrictive than the previous one, all the lower bounds obtained for MOD1 are still valid in MOD2.

## 2.2. Coding information

The information at a point is given by the value of an electrical parameter at this point, which we define as the potential of a capacitor. While electrical computations are essentially analog processes, the coding of information is made digital by assuming a 0 for a potential less than  $V_0$  and a 1 for a potential greater than  $V_1$  ( $V_1 > V_0$ ).

## 2.3. Wires

A wire is a rectangular parallelepiped made of conducting material, oriented by the direction of the current. It is characterized by its length  $L$ , its width  $W$ , its thickness  $H$ , and its distance  $D$  from a plane of reference (the substrate). Its resistance  $R$  and its capacitance  $C$  are given by the (idealized) relations

$$R = \rho \times L / (W \times H) \quad C = \epsilon \times L \times W / D$$

where  $\rho$  and  $\epsilon$  are technology-dependent coefficients.

Minimum values for  $L, W$  and  $H$  are set by the technology (as well as by the laws of physics), and we

require  $D$  to be constant for any wire. Moreover it is legitimate always to assume bounded thickness. Indeed a current density  $\delta$  causes a heat power loss in the wire proportional to  $L \times W \times H \times \delta^2$ , but the dissipated power is proportional to  $L \times W$ , since the circuit is planar. For allowing this heat to be dissipated, the thickness  $H$  must remain within constant bounds. Thus we can assume that the resistance is simply proportional to  $L/W$  and the capacitance to  $L \times W$ .

## 2.4. Nodes

We distinguish three kinds of nodes, each of which uses up a minimum constant area.

- *Connection nodes*: Their purpose is to provide electrical contacts between a bounded number of wires. These contacts may either connect wires on a same layer, or they may be "vertical contacts" between different layers. Of course, they introduce no delay and do not dissipate any energy.
- *I/O ports*: They ensure the exchange of information between the circuit and the outside world. The locations and the order in which input (resp. output) bits are to be written (resp. read) are fixed and independent of the values of these bits. We restrict each input bit to be available on the input port only once. This implies that the repeated use of the same input bit necessitates its storage within the circuit. The transmission of an information signal through an I/O port introduces a constant delay.
- *Gates*: Conceptually, a gate is the device used to compute a logical function of one or two inputs and one output. Since it can be shown that there is no interest in having gates of arbitrary size, we assume that all gates have the same size. Physically we must associate a gate capacitance with each input. An input is valid as soon as the corresponding gate capacitor has been set above or below a certain threshold potential. The value of the function is given by the potential of the output device of the gate. Once the output is available, it cannot be destroyed before a constant lapse of time, whatever the input changes occurred in the meantime.

## 2.5. Current density

**Proposition 1:** The density of current is bounded at any point of a conductor by a maximum value  $\delta_{\max}$ .

One major flaw in previous models is to suppose that a wire of constant width can drive a current of arbitrary intensity. We can list at least three reasons in present-day technologies which justify Proposition 1.

1. Any conductor with non-zero resistance produces a power per unit volume proportional to the *square* of the current density. Since this power can be dissipated only through the boundary of the conductor, the heat dissipation is at most proportional to the area of the conductor, which implies a bounded density.
2. An electrical phenomenon called *metal migration* [MC80, CL80] causes a current to destroy the conductor all the more quickly as the density is high. For this reason, a maximum admissible density of current can be assigned to any conducting material.
3. The voltage drop per unit length is proportional to the density of the current. Since we must ensure that the logical value of the signal provided by power wires is the same at any point of the circuit, this voltage drop must remain small, and thus the density must be bounded.

For example, the aluminium currently used in NMOS technology has a maximum density imposed by metal

migration of about  $10^9 \text{ A/m}^2$  or only  $1 \text{ mA}/\mu\text{m}^2$ . For this density the voltage drop is  $30 \text{ V/m}$  with a resistivity of about  $3 \cdot 10^{-8} \Omega \times \text{m}$ . Note that the voltage drop on a  $3 \text{ mm}$  wire is  $0.1 \text{ V}$ , and is far from negligible. Also, the power induced in the wire by such a density of current is about  $3 \text{ W/cm}^2$ , if the thickness is  $1 \mu\text{m}$ .

### 3. Transmitting Information

We turn to the problem of transmitting an information bit from a point A to a point B at a (Euclidian) distance  $L$  apart. We will assume that this information will be carried through an arbitrary path from A to B consisting of nodes and wires.

We first consider the case where the path consists of a wire followed by a gate. Let  $S = H \times W$  be the section of the wire. In order to transmit a bit of information, we must raise the wire to the required voltage. The charge  $Q$  on the wire is therefore proportional to its capacitance, that is,  $L \times W$ . Since in a time  $T$  a density of current crossing a section  $S$  can provide at most a charge  $\delta_{\max} \times S \times T$ , the assumption that  $H$  is bounded yields the relation  $T = \Omega(L)$ .

We next investigate the case where two paths of the previous type are cascaded. Since the first gate cannot be switched before the signal becomes available on the first wire, the total delay will amount to the added delays of the two paths augmented by the switching time of the first gate. This also results in an  $\Omega(L)$  delay. The last case to examine involves two wires linked by a connection node. We can apply the reasoning used in the case of a single wire, with  $W$  now being the maximum of the two wire widths. The same result follows directly. In the general case, we can decompose an arbitrary path from A to B into components of the form previously examined. Putting the above results together permits us to find the claimed lower bound on the time.

In addition, we should notice that some energy is dissipated along the wire during the propagation of information since the wire has a non-zero resistance. This energy is proportional to the charge involved, which is  $\Omega(L)$  in any configuration. Observe that this energy is independent of the time  $T$ .

Both results permit us to state the following.

**Theorem 2:** Transmitting a signal between two points at a distance  $L$  apart requires  $\Omega(L)$  time and  $\Omega(L)$  energy.

Note that this lower bound cannot be achieved with a simple wire: because of the diffusion law [MC80,SE79], the actual delay is in fact proportional to  $R \times C = L^2$ . However, we can reduce this delay to  $O(L)$  by using  $O(L)$  wires of constant length connected by  $O(L)$  gates (e.g., inverters or amplifiers). If the wires have minimum width, the lower bound  $O(L)$  on the energy is also achieved. Note that a simple speed-of-light argument yields the same result for any technology. This is precisely what makes MOD1 a minimal planar model for all physical computations.

#### 4. Distributing and Collecting Information

Throughout this section, we will assume that the model is MOD1 or MOD2, indifferently. To fan-in or fan-out information being two of the most common operations performed by circuits, we next turn to these problems, from which we can best measure the significant departure of our models from previous ones. For simplicity, we first prove a technical lemma.

**Lemma 3:** There is a constant  $c$  ( $c = 1/2\pi$ ) such that for any convex polygon with a boundary of length  $N$  and for any point  $M$ , there exists a vertex  $v$  such that  $\text{dist}(v, M) \geq cN$ .

We omit the proof, which is straightforward.

##### 4.1. Fan-out

A fan-out of degree  $N$  refers to the distribution of an information bit from a source to  $N$  points (gates or ports) on the circuit. To study the complexity of this problem, we distinguish two cases; when the  $N$  points lie on a convex boundary (e.g., on the boundary of the circuit), and when their location is left arbitrary. We define  $T$  (resp.  $E$ ) to be the minimum time (resp. energy) to perform a fanout of  $N$  points. It is trivial to see that  $E = \Omega(N)$  in both cases, since to reach every node, the information must cross a wire of (at least) unit length. As for the time  $T$ , we have two different results.

**Theorem 4:** If the  $N$  points lie on a convex boundary,  $T = \Omega(N)$ .

**Proof:** It follows from Lemma 3 that one of the  $N$  destinations is at least  $cN$  apart from the source, and Theorem 2 permits us to conclude.  $\square$

**Theorem 5:** If the  $N$  points have arbitrary locations,  $T = \Omega(N^{1/2})$ .

**Proof:** A consequence of the fact that the maximum distance between  $N$  points and an arbitrary point in the plane is at least  $cN^{1/2}$ , for some constant  $c$ .  $\square$

Note that all these lower bounds are tight, as shown in Figure 4-1.

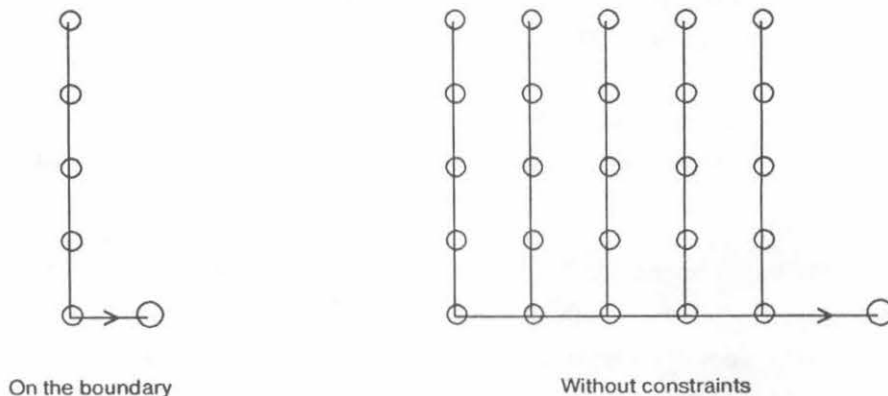


Figure 4-1: Optimal fan-out.

## 4.2. Fan-in

The fan-in is essentially the reverse operation of the fan-out, since  $N$  information bits must converge from  $N$  sources to one destination point. Yet it is a little more general, since the information may be submitted to logical operations on its way to the destination. Typically, the problem is to compute a boolean function of  $N$  inputs and one output. Since every gate is followed by a wire of unit length at least, the minimum energy dissipated during the operation is  $E = \Omega(N)$ . If the  $N$  inputs are valid at the same time, the results are the same as for the fan-out. In the more general case where pipelining is allowed and the inputs are valid at arbitrary times, we can show the following.

**Theorem 6:** If  $T$  (resp.  $A$ ) denotes the minimum time (resp. area) for computing a boolean function of  $N$  inputs, we have  $T = \Omega(N^{1/2})$  and  $AT = \Omega(N)$ .

**Proof:** Let  $p$  denote the total number of input ports actually used. It takes time at least proportional to  $N/p$  to read all the inputs, and since the  $p$  ports lie on a convex boundary, Lemma 3 and Theorem 2 show that  $T = \Omega(p)$ . Observing that  $A = \Omega(p)$ , the result is then immediate.  $\square$

Note that these lower bounds are still valid for boolean functions with an arbitrary number of outputs, as long as at least one output depends on all the input values. The addition for example falls in that category, since the last carry depends upon all the operand bits. If the boolean function is a commutative, associative, operation on  $N$  variables, these lower bounds are tight, as shown in Figure 4-2.

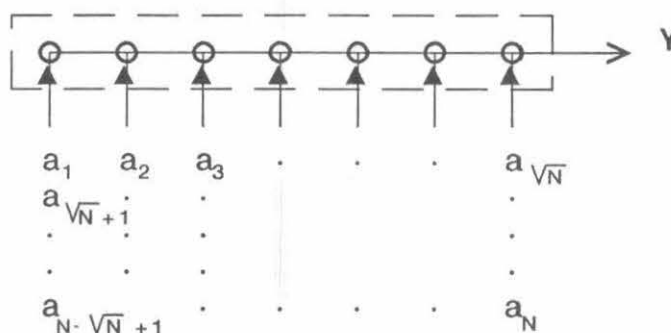


Figure 4-2: Computing  $Y = a_1 \text{ op } a_2 \dots \text{ op } a_N$  takes  $\Omega(N^{1/2})$  time and area.

## 5. New Lower Bounds for some Common Problems

### 5.1. Addition

Since our models relate the time of computation to the geometry rather than the topology of the circuit, we can show that many complete binary tree based schemes cease to have the logarithmic time complexity which they enjoy in previous models. Notable examples include the fan-in and fan-out operations studied earlier, or the addition of two  $N$ -bit integers, to which we next turn our attention. We study this problem in our two

models in turn. For simplicity, we start the analysis with the model MOD2, and present the basic arguments.

### 5.1.1. Case MOD2

**Theorem 7:** If  $T$  is the time required in MOD2 by any circuit to add two  $N$ -bit integers, and if  $A$  is the area of the circuit, we have

$$1) AT = \Omega(N) \quad 2) T = \Omega(N^{2/3}).$$

**Proof:** For the sake of simplicity, we will assume in this proof that the sign "=" really means "equals to within a constant factor". Relation 1) follows directly from the fact that adding two  $N$ -bit integers involves a fan-in of degree  $N$ . To prove 2), let's call  $X$  one of the operands and  $Y$  the result of the addition. Since we can always assume that low order bits are read first, we can rewrite  $X$  as  $X_p \dots X_2 X_1$ , where  $X_i$  are the bits of  $X$  read at time  $t_i$ , with  $t_1 < \dots < t_p < T$ .  $X_i$  denoting both the chain of bits and its length, we have the relations

$$(1) X_1 + \dots + X_p = N \quad (2) T \geq p.$$

Let  $X(t)$  be the total number of bits of  $X$  read so far at time  $t$ , and let  $Y(t)$  be the total number of result bits output in this interval of time. Since at time  $t$ , the total number of possible values for the remaining output bits is at least  $2^{N-Y(t)}$ , and only  $N-X(t)$  bits of  $X$  remain to be read, the circuit must have at least  $X(t)-Y(t)$  active gates at time  $t$ . This requires a circuit perimeter  $\Pi \geq X(t)-Y(t)$  and a time  $\Pi$ , hence the relations

$$(3) X(t)-Y(t) \leq \Pi \quad (4) T \geq \Pi.$$

Since low-order input bits are read first and a fan-in on  $k$  bits takes  $\Omega(k)$  time, at least  $N-X(t_i)$  output bits remain to be computed at time  $t_i + X_i$ . We can give a geometric interpretation of this relation as shown in Figure 5-1. Relation (3) implies that the endpoints of the intersection of the shaded area of Fig.5-1 with a vertical line are at most  $\Pi$  apart. It follows that the shaded area must lie within the strip  $(L_1, L_2)$ , which in turn implies

$$X_1^2 + \dots + X_p^2 \leq T\Pi$$

Since  $X_1^2 + \dots + X_p^2$  is minimal when all the  $X_i$ 's are equal, we derive the relation  $N^2/p \leq T\Pi$ , which combined with relations (2) and (4) yields

$$T \geq N^2/p\Pi + p + \Pi$$

The minimum of the right-hand side is achieved for  $p = \Pi = N^{2/3}$ , which concludes the proof.  $\square$

Note that the lower bound on  $AT$  is trivially tight, since there exist linear-time constant-area adders. We do not believe that this is the case with the lower bound given for  $T$ . We conjecture that  $T = \Omega(N)$  is the actual lower bound in this model, which would make the simplest adder in the world asymptotically optimal.

### 5.1.2. Case MOD1

It is natural that lower bounds obtained in MOD1 should be weaker than in MOD2. However, MOD1 has the merit of greater generality, and any lower bound in this model is thus very interesting.

**Theorem 8:** If  $T$  is the time required in MOD1 by any circuit to add two  $N$ -bit integers, and if  $A$  is the area of the circuit, we have

$$T = \Omega(N^{1/2}), \quad AT = \Omega(N), \quad AT^2 = \Omega(N^2).$$

**Proof:** The first two relations result from the fact that adding two  $N$ -bit integers involves a fan-in of degree  $N$ . Indeed the last carry is a fan-in of all the input bits. We can prove the last relation with the same technique used above. Keeping the same notation, we find that  $X(t)-Y(t) \leq A$ , since at any time  $t$  the number of bits stored in the circuit is at least  $X(t)-Y(t)$ . On the other hand,  $Y(t)$  always lies below the shaded area of Fig.5-1. It then follows that total area of the shaded region

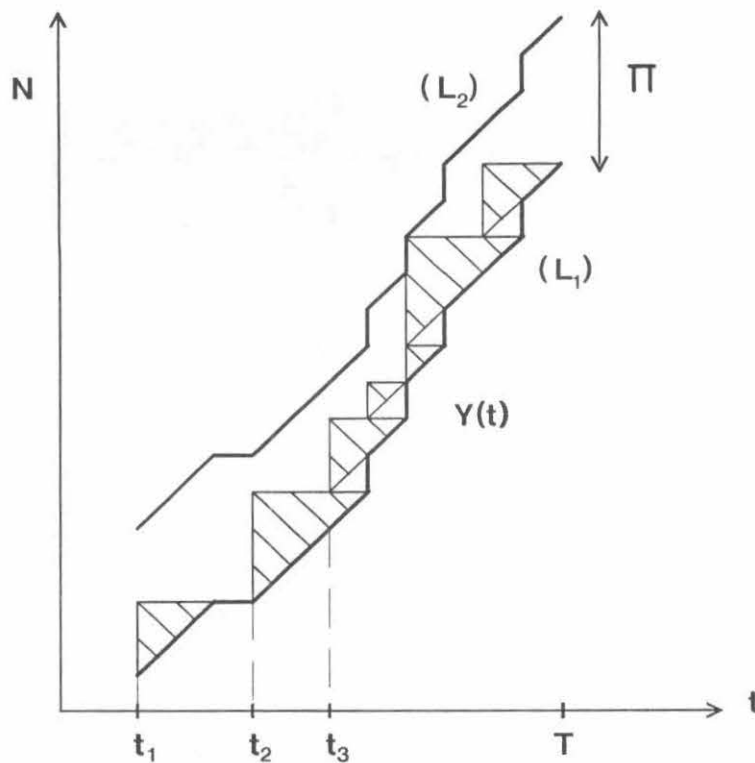


Figure 5-1: The  $\Omega(N^{2/3})$  time lower bound on integer addition.

cannot exceed the area of the parallel strip  $(L_1, L_2)$ , hence

$$X_1^2 + \dots + X_p^2 \leq AT.$$

The minimum is achieved for  $X_i = N/p$ , and since the time for reading the data is proportional to  $p$ , we find  $AT^2 = \Omega(N^2)$ , which completes the proof.  $\square$

### 5.1.3. Optimal adders in model MOD1

A fortunate feature of addition in model MOD1 is to allow the possibility of matching all the lower bounds derived above. We will describe a class of adders which satisfy these properties.

**Serial Adder:** The simplest adder requires constant area, operates in linear time, and thus matches the lower bound for the measures  $AT$  and  $AT^2$ . The scheme of this adder is represented in Fig.5-2.

**CLA Adder:** Assuming wlog that  $N$  is a power of two, we implement the CLA scheme on a complete binary tree with  $N$  leaves. The operand bits are read in parallel at the leaves, and the time of computation is at least the time for propagating a signal across the longest path in the tree. It follows that the layout of Fig.5-3 requires  $O(N)$  time and  $O(N \log N)$  area.



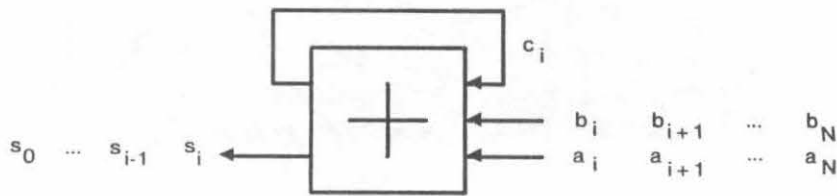


Figure 5-2: The Serial Adder.

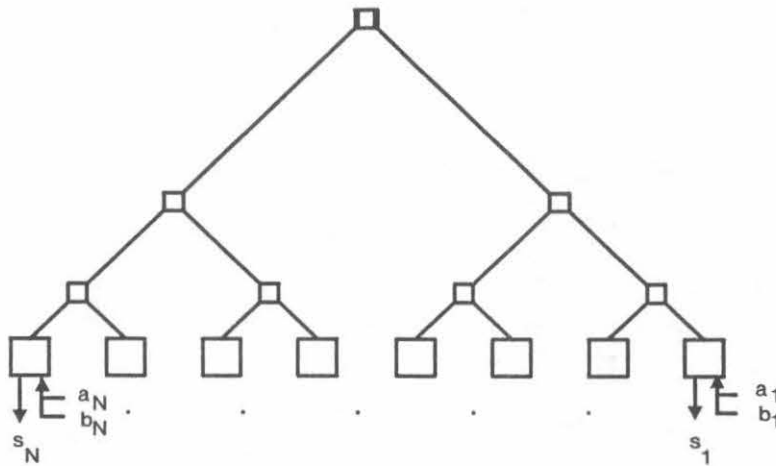


Figure 5-3: The CLA Adder.

If the technology allows the packing of  $k$  information bits on a square of area  $O(k)$  (e.g. that excludes NMOS), an alternate layout may use the H-embedding of a binary tree, as shown in Fig.5-4. The operands may be driven from the input ports to the leaves of the tree in about  $N^{1/2}$  waves of  $2N^{1/2}$  bits. Unfortunately, each wave consists of a complicated (but fixed) sequence of input bits. If we do not account for the task which arranges the input bits in the proper order, and if we use inverters to avoid long wires (see Section 3) adding two  $N$ -bit integers simply takes  $O(N^{1/2})$  time and linear area, which matches the lower bounds obtained for  $T$  and  $AT^2$ .

**Mixed CLA Adder:** In some applications the size of the operands greatly exceeds that of the circuit, and only, say,  $N^\alpha$  input ports are available. In this case, we can divide the operands into roughly  $N^{1-2\alpha}$  groups of  $N^{2\alpha}$  bits, and compute the addition for each group with a CLA adder of area  $N^{2\alpha}$ , transmitting the carry for the next addition every time around. The total time of computation will thus be  $O(N^{1-\alpha})$ , with a circuit of area  $O(N^{2\alpha})$ . Note that the lower bound  $AT^2 = N^2$  is still matched with this scheme. Also, we observe that for  $\alpha = 1/2$  we have the CLA adder, whereas setting  $\alpha$  to zero reduces to the serial adder.



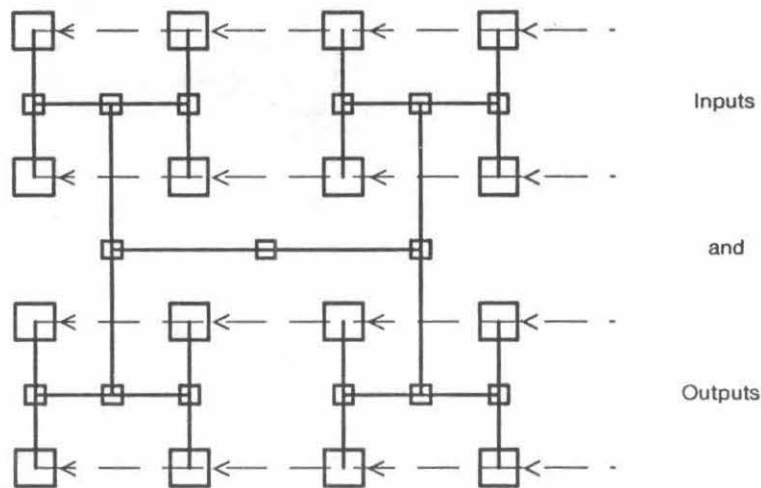


Figure 5-4: Optimal layout of the CLA adder.

## 5.2. Transitive Functions

In a recent paper [VU80], J. Vuillemin has shown that the transitivity of a function has heavy consequences on its complexity in a VLSI model. Roughly speaking, a function is said to be transitive of degree  $N$  if it computes a transitive group of permutations acting on  $N$  elements. This implies that the function can map any input bit onto any output bit for an appropriate value of the other inputs. Such functions include cyclic shifts, integer products, convolutions, linear transforms, and some matrix products.

### 5.2.1. Case MOD1

Even in our more general model, we can show a significant difference with previous results [PV79, VU80].

**Theorem 9:** Computing a transitive function of degree  $N$  takes time  $T = \Omega(N^{1/2})$ .

**Proof:** Let  $p$  be the number of output ports actually used. Since an input bit can be mapped onto any output port, Lemma 10 shows that for some value of the inputs, the computation will take time at least proportional to  $p$ . On the other hand, observing that it takes time at least proportional to  $N/p$  to output the result completes the proof.  $\square$

It is worthwhile to notice the serious gap existing between this model and the previous ones, which allowed for logarithmic time for computing transitive functions (e.g. the CCC-scheme [PV79]).

### 5.2.2. Case MOD2

It comes as no surprise that since our second model adds physical constraints to the one in which Vuillemin derived his lower bounds, we can significantly improve upon his results. Before proceeding, we will establish a preliminary result.

**Lemma 10:** If  $N$  gates in a circuit are switched at the same time, their convex hull has a perimeter  $\Omega(N)$ .

**Proof:** Since all the power comes from outside the circuit and is transmitted through wires, the power inside any convex region of the circuit is at most proportional to its perimeter. Switching a gate requiring a minimum energy, the result is straightforward.  $\square$

We can now prove our main result.

**Theorem 11:** Any circuit of area  $A$  which computes a transitive function of degree  $N$  in time  $T$  satisfies  $A = \Omega(N)$ ,  $T = \Omega(N)$ .

**Proof:** It has been shown in [VU80] that the circuit must have the capability of memorizing  $N$  bits. Therefore Lemma 10 implies that the circuit must have two active gates  $G_1$  and  $G_2$  at a distance  $\Omega(N)$  apart, hence  $A = \Omega(N)$ . We can always assume that for some values of the inputs, information will be transmitted from  $G_1$  to an output port  $P_1$  (same with  $G_2$  and an output port  $P_2$ ). Consider now an arbitrary input port  $R$ . Since the function is transitive, there exists a path in the circuit from  $R$  to  $P_1$  and from  $R$  to  $P_2$ . Among all possible computations, the four paths  $G_1$ - $P_1$ ,  $G_2$ - $P_2$ ,  $R$ - $P_1$ , and  $R$ - $P_2$  will be used at least once. From Theorem 2, it then follows that  $T$  is at least proportional to  $\text{Max}\{G_1P_1, G_2P_2, RP_1, RP_2\}$ . The sum of these four lengths is greater than  $G_1G_2 = \Omega(N)$  -See Fig.5-5-, which concludes the proof.  $\square$

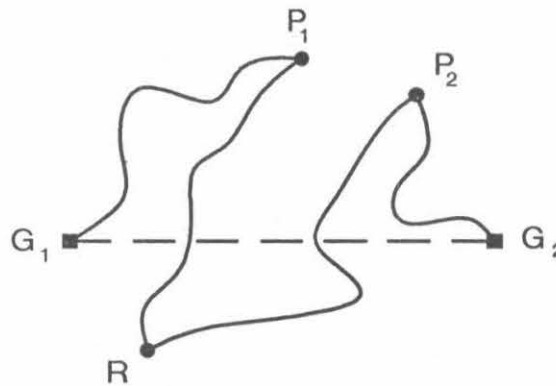


Figure 5-5: Computing a transitive function requires linear time.

Remark: In MOD2, these lower bounds are tight for some problems; for example optimal circuits for performing integer multiplication, based on the Shift&Add scheme, can be found.

## 6. Conclusions

The major contribution of this paper has been to show how previous models fail to allow for asymptotic analysis. We have proposed two models of computation which are more realistic yet fairly simple. Since our models are essentially geared towards asymptotic analysis, previous models may turn out to be more accurate for circuits of small size. For example, the carry-look-ahead scheme for adding two  $N$ -bit integers actually requires at least  $\Omega(N^{1/2})$  time in our models instead of the well-known logarithmic time, but it may still be superior to any naive circuit for small integers.

Further refinements of these models should be valid independently of size considerations, and should allow for *à la Knuth* analyses of VLSI circuits. It is still difficult to think of a technology-independent model at the present time. But it may be a prerequisite for building a complexity theory which faithfully reflects reality.

## Acknowledgments

We wish to thank Jean Vuillemin for suggesting this research and Mike Foster for many fruitful discussions. Our thanks also go to H.T. Kung and Gerard Baudet, who shared our interest in this work.

## References

- [BK80] R.P. Brent and H.T. Kung, *The Chip Complexity of Binary Arithmetic*, proc. 12th Annual ACM Symposium on Theory of Computing, ACM, pp. 190-200, May 1980.
- [CL80] W.A. Clark, *From Electron Mobility to Logical Structure: A View of Integrated Circuits*, Computing Surveys, Vol.12, No 3, September 1980.
- [MC80] C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980.
- [PV79] F.P. Preparata and J. Vuillemin, *The Cube-Connected-Cycles: A Versatile Network for Parallel Computation*, proc. 20th Annual Symposium on Foundations of Computer Science, Oct. 1979.
- [SE79] C.L. Seitz, *Self-timed VLSI Systems*, proc. of Caltech Conf. on VLSI, 1979.
- [TH79] C.D. Thompson, *Area-Time Complexity for VLSI*, proc. 11th Annual ACM Symposium on Theory of Computing, ACM, pp. 81-88, May 1979.
- [VU80] J. Vuillemin, *A Combinatorial Limit to the Computing Power of V.L.S.I. Circuits*, proc. 21st Annual Symposium on Foundations of Computer Science, Oct. 1980.



## A LOGIC DESIGN THEORY FOR VLSI\*

by

John P. Hayes

Digital Integrated Systems Center

and

Departments of Electrical Engineering and Computer Science

University of Southern California

Los Angeles, California 90007

ABSTRACT

Classical switching theory fails to account for some key structural and logical properties of the transistor circuits used in VLSI design. This paper proposes a new logic design methodology called CSA theory which is suitable for VLSI. Three kinds of primitive logic devices are defined: connectors (C), switches (S), and attenuators (A); the latter have the characteristics of pullup/pulldown components. It is shown that four new logic values are required, in addition to the usual Boolean 0 and 1 values. These values introduce a concept of gain or drive capability into logic design; they also account for the high-impedance state of tri-state devices. The elements of CSA theory and its application to some basic VLSI design problems are described. It is demonstrated that CSA theory provides a more powerful and more rigorous replacement for the mixed logic/electronic methods currently used in VLSI design.

---

\*This research was supported by the National Science Foundation under Grant No. MCS78-26153, and by the Naval Electronic Systems Command under Contract No. N00039-80-C-0641.

## 1. INTRODUCTION

The development of very large-scale integrated (VLSI) circuits using the philosophy espoused by Mead and Conway [1] and others involves a complex interplay of various design techniques at the electronic, logical and systems levels. These techniques are ad hoc for the most part, with the result that the VLSI designer is mainly guided by experience rather than theory. It might be expected that the large body of results in switching theory and logic design that has accumulated over the past 40 years can readily be applied to VLSI design, at least for analysis purposes if not for synthesis. This does not appear to be the case, however. Several reasons may be cited for this.

(1) The basic component of VLSI circuits is the MOS transistor whose logical behavior is that of a three-terminal digital switch. Neither of the classical models from switching theory, branch-type networks (also called contact networks) or gate-type networks [2], adequately capture the structure or logical behavior of MOS transistor circuits. The primitive components of gate-type circuits are logic gates which allow signal transmission in one direction only. An MOS transistor, on the other hand, is inherently bidirectional. The components of branch-type networks are (relay) contacts. A contact is bidirectional, but unlike a transistor, it is basically a two-terminal device.

(2) Classical switching theory hides some types of logic devices that have a significant impact on integrated circuit design and layout. For example, it does not recognize the important role played by connectors in logical behavior. Connections to power and ground are omitted from standard logic diagrams, yet they are the sources of the logical 0 and 1 values on which the logical operation of all circuits depends. The selection and layout of connectors is a central issue in VLSI design. Components like amplifiers and pullup/pulldown loads, which are crucial to proper logical or digital operation, are also invisible at the standard logic level. To see these devices we must move to the more detailed electronic or analogue level.

(3) Only the two logical values 0 and 1 are recognized in standard switching theory. However, in modern design practice extensive use is made of

at least one additional logic value, the high-impedance state  $Z$ . Indeed it has been suggested that MOS technology is inherently a three-state technology [3].

At the present time, the usual remedy for the foregoing difficulties is to combine design methods from switching theory and electronic circuit theory heuristically. This results in "mixed" circuit diagrams which couple logic gates, transistors, etc. in a manner that, strictly speaking, is meaningless. It also causes important logic design techniques such as wired logic and tri-state logic to be treated as anomalous special cases.

In this paper a new logic design theory is introduced that attempts to overcome the difficulties cited above. The key components of this theory are connectors (C), switches (S) and attenuators (A); we therefore refer to it as *CSA theory*. A switch here is a three-terminal device that can accurately model the digital operation of a PMOS or NMOS transistor. An attenuator models a pullup or pulldown load; it has no counterpart in classical switching theory. Central to our approach is a six-valued logic which, in addition to the usual "strong" Boolean values 0 and 1, has "weak" versions of these values denoted by  $\tilde{0}$  and  $\tilde{1}$ . This weak/strong signal dichotomy allows the electronic concepts of signal amplification and attenuation to be transferred to the logic level. The high-impedance state  $Z$  is also treated as an explicit logic value. CSA theory provides a uniform and consistent alternative to the mixed design approach mentioned earlier. It can be used to analyze both branch- and gate-type networks, as well as such nonclassical structures as wired logic and tri-state logic.

Section 2 presents an informal development of the basic concepts of CSA theory. In Sec. 3 these ideas are presented in more rigorous and complete fashion. Finally in Sec. 4, CSA theory is applied to the design of a simple but important class of circuits, namely inverters.

## 2. INFORMAL DEVELOPMENT

In this section the basic concepts of connector and switch are examined in detail. We show that six logical values are needed for an adequate description of their behavior, as well as a new logic device which we call an attenuator.

### Connectors

In classical switching theory the only logical operation associated with a connector or wire is the trivial identify function  $v \rightarrow v$ . At the CSA level of complexity, connectors are seen as the fundamental devices for performing nontrivial operations of the AND and OR type. To demonstrate this, we first need a precise definition of a connector and its behavior. A *terminal*  $T$  is a designated connection point in a network; it is denoted by a black dot in logic diagrams as shown in Fig. 1a. A *simple connector* is a continuous conducting path between two terminals. It may represent a metal, diffusion or polysilicon conductor in an integrated circuit, and is represented by a line as in Fig. 1b. A (complex) *connector* is a linked set of simple connectors; Fig. 1c shows an example. Any point in a connector may be designated a terminal, therefore a connector can be viewed as simply a sequence of contiguous terminals.

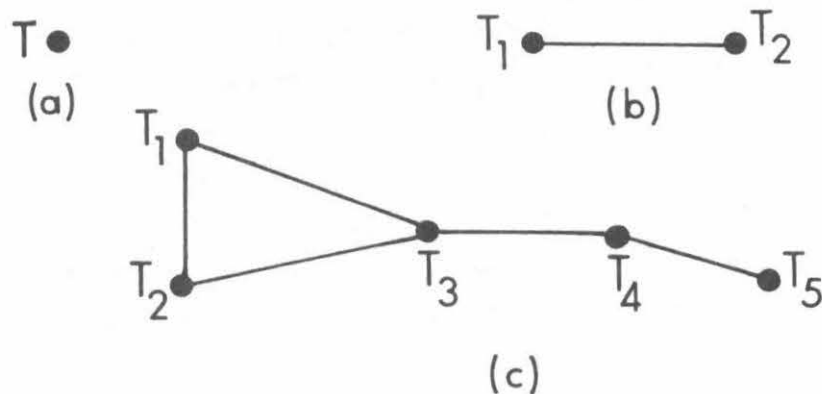


Fig. 1 (a) A terminal  $T$ . (b) A simple connector. (c) A general connector  $C$ .

Let  $V$  be the set of logic values or signals of interest.  $V$  contains the usual Boolean constants 0 and 1; additional values will be added later. With every connector  $C$  we associate a set of *input values*  $v_{in}(C)$  taken from  $V$ . The  $v_{in}(C)$  values are typically derived from external signal sources that are connected to  $C$ . Thus in the connector of Fig. 2a the external signal sources are indicated by arrows, and the input signal set is  $v_{in}(C) = \{v_1, v_2, v_3, v_4, v_5, v_6\}$ .



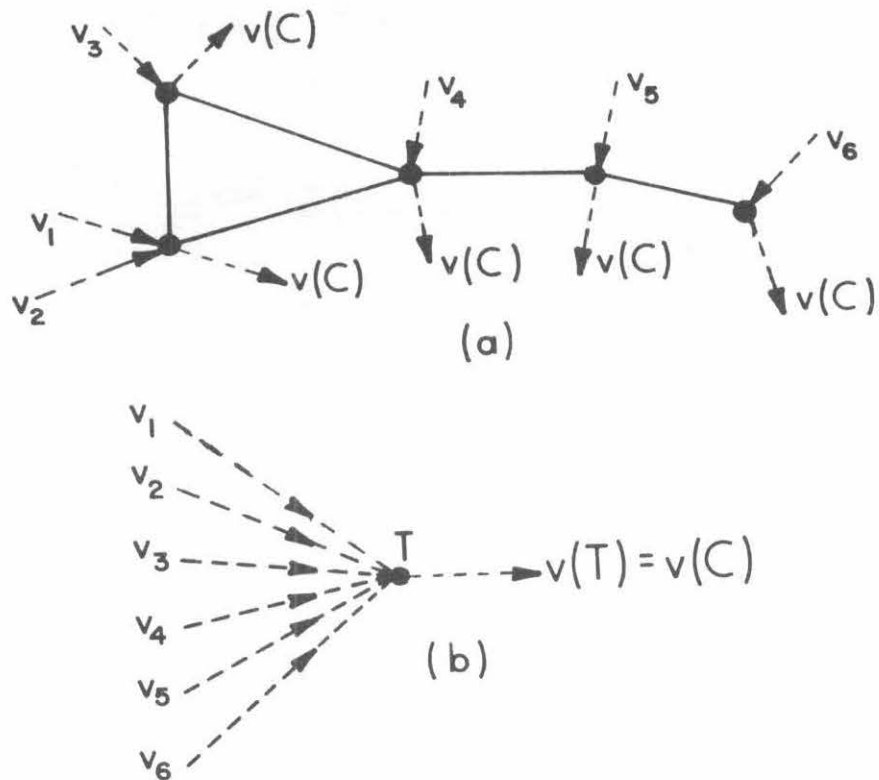


Fig. 2 (a) Input-output signals of the connector  $C$ . (b) An equivalent terminal  $T$ .

While several different input values may be applied to a connector simultaneously, we assume that the connector produces a unique *output value*  $v(C)$  at all its terminals, where  $v(C) \in V$ . Thus if the physical signals associated with  $C$  are voltages, then  $C$  has the equipotential property of a perfect electrical conductor. It follows that a complex connector can always be replaced by a single terminal as illustrated in Fig. 2b.

Suppose that the input values  $v_1, v_2 \in V$  are applied to connector  $C$ . For logical consistency and completeness, we require  $v(C)$  to be defined uniquely for all possible combinations of  $v_1$  and  $v_2$ . We can write

$$v(C) = \#(v_1, v_2)$$

where  $\#$  denotes the *connection function* implemented by  $C$ . Let  $v_1$  and  $v_2$  assume the values 0 and 1. If  $v_1 = v_2$ , then we expect the following equations to hold:

$$\begin{aligned}\#(0,0) &= 0 \\ \#(1,1) &= 1\end{aligned}\tag{1}$$

If  $v_1 \neq v_2$  is allowed (this is normally considered to be improper behavior in binary switching circuits), then we need a third logic value which we denote  $U$ .  $U$  (for unknown) has frequently been used in logic simulation programs to model signal values during transitions between 0 and 1, and the values associated with uninitialized states [4]. We use  $U$  in the sense of a conflict value that results in the connector behavior defined by the following set of equations:

$$\#(0,1) = \#(0,U) = \#(1,U) = \#(U,U) = U\tag{2}$$

Next we consider the notion of a switch as a controlled connector, and show that it requires the introduction of three additional logic values.

### Switches

A *switch*  $S$  is defined here as a three-terminal device with a "control" terminal  $K$  and two symmetric "data" terminals  $D_1$  and  $D_2$ . It is represented by the circuit symbol of Fig. 3a. The set of values  $V(D)$  is assigned to  $D_1$  and  $D_2$ . The set  $V(K)$  containing the two values ON and OFF is assigned to  $K$ . Later we will equate  $V(K)$  and  $V(D)$ . When  $v(K) = \text{ON}$ ,  $D_1$  and  $D_2$  are joined by a connector as in Fig. 3b. When  $v(K) = \text{OFF}$ , there is no connection between  $D_1$  and  $D_2$  via the switch. Examples of switches that can be easily made to conform to this model are a manual on-off switch, a single-contact relay, and an MOS transistor.

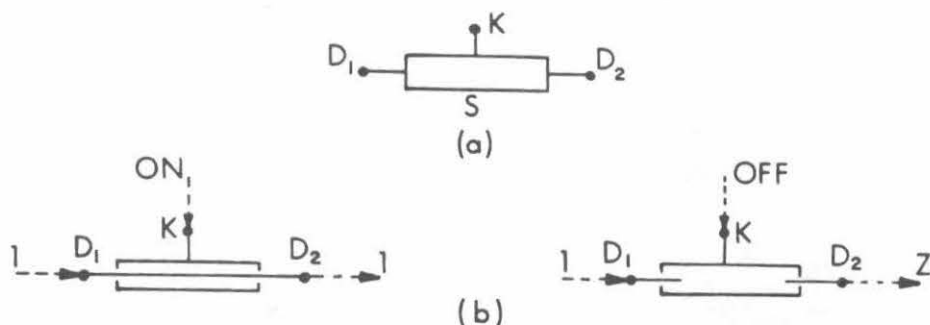


Fig. 3 (a) An isolated switch  $S$ , and (b) its behavior.

Suppose that an isolated switch  $S$  is to be used to control the signal value appearing at one of its data terminals, say  $D_2$ . Intuitively the following type of behavior is expected:

$$v(K) = \text{ON} \quad \text{implies} \quad v(D_2) = 1 \quad (3)$$

$$v(K) = \text{OFF} \quad \text{implies} \quad v(D_2) = 0 \quad (4)$$

If  $v(K) = \text{ON}$ , then  $v(D_1) = v(D_2)$ , so we can satisfy (3) by applying the constant 1 to  $D_1$  as shown in Fig. 3b. When  $v(K) = \text{OFF}$ , however,  $D_2$  becomes an isolated terminal, and it "floats" to a value that is distinct from 0, 1 and  $U$ . We therefore introduce a new logical value  $Z$  to denote  $v(C)$  when  $C$  is an isolated connector.  $Z$  corresponds to the usual high-impedance state of tri-state logic. It is a weak value in the sense that it can be overridden by each of the logic values 0, 1 and  $U$ . This suggests that  $Z$  should satisfy the following set of equations:

$$\begin{aligned} \#(0, Z) &= 0 \\ \#(1, Z) &= 1 \\ \#(U, Z) &= U \\ \#(Z, Z) &= Z \end{aligned} \quad (5)$$

To satisfy (4) above, we can attempt to apply to  $D_1$  an external signal  $v$  that forces  $v(D_2)$  to 0. Thus when  $v(K) = \text{OFF}$ , we require

$$\#(v, Z) = 0 \quad (6)$$

To satisfy (3) at the same time requires

$$\#(v, 1) = 1 \quad (7)$$

assuming that 1 has been applied to the input data terminal  $D_1$ . It is easily seen that none of the values 0, 1,  $U$ ,  $Z$  can satisfy equations (1), (2), (5), (6) and (7) simultaneously. Thus we introduce a fifth logical value denoted

$\tilde{0}$  which, like 0, is an acceptable "0-like" value in the realization of Boolean functions. If we replace (6) and (7) by

$$\#(\tilde{0}, Z) = \tilde{0}$$

and

$$\#(\tilde{0}, 1) = 1 \quad (8)$$

respectively, no contradiction results. Now (8) implies that 1 overrides  $\tilde{0}$  when both are applied to the same connector, hence  $\tilde{0}$  is a weak 0-like value, that is, a value with low (logical) drive capability. In a similar manner, we define a weak 1-like value denoted  $\tilde{1}$ . The foregoing analysis suggests that  $\tilde{0}$  and  $\tilde{1}$  should satisfy the following set of equations involving the connection operator  $\#$ :

$$\begin{aligned} \#(\tilde{0}, Z) &= \#(\tilde{0}, \tilde{0}) = \tilde{0} \\ \#(\tilde{1}, Z) &= \#(\tilde{1}, \tilde{1}) = \tilde{1} \\ \#(\tilde{0}, 0) &= \#(\tilde{1}, 0) = 0 \\ \#(\tilde{0}, 1) &= \#(\tilde{1}, 1) = 1 \\ \#(\tilde{0}, \tilde{1}) &= \#(\tilde{0}, U) = \#(\tilde{1}, U) = U \end{aligned} \quad (9)$$

### Attenuators

We have just seen that if a switch is used to transmit 0-like and 1-like signals, we need two new values  $\tilde{0}$  and  $\tilde{1}$  that can be applied externally to its data terminals. We now define a new logic element called an attenuator whose function is to generate  $\tilde{0}$  and  $\tilde{1}$ . An *attenuator* is a unidirectional two-terminal device whose output is  $\tilde{0}$  or  $\tilde{1}$  when 0 or 1 respectively are applied to its input terminal. Figure 4 shows the symbol used for an attenuator, as well as its typical use to force the output of a switch to have values from the set  $\{0, 1, \tilde{0}, \tilde{1}\}$ . The circuit of Fig. 4 is thus a complete switching circuit that meets the original behavior specifications suggested by (3) and (4).

It is apparent from Fig. 4 that an attenuator is a device that can pull an isolated connector up (from Z to  $\tilde{1}$ ) or down (from Z to  $\tilde{0}$ ). It thus models

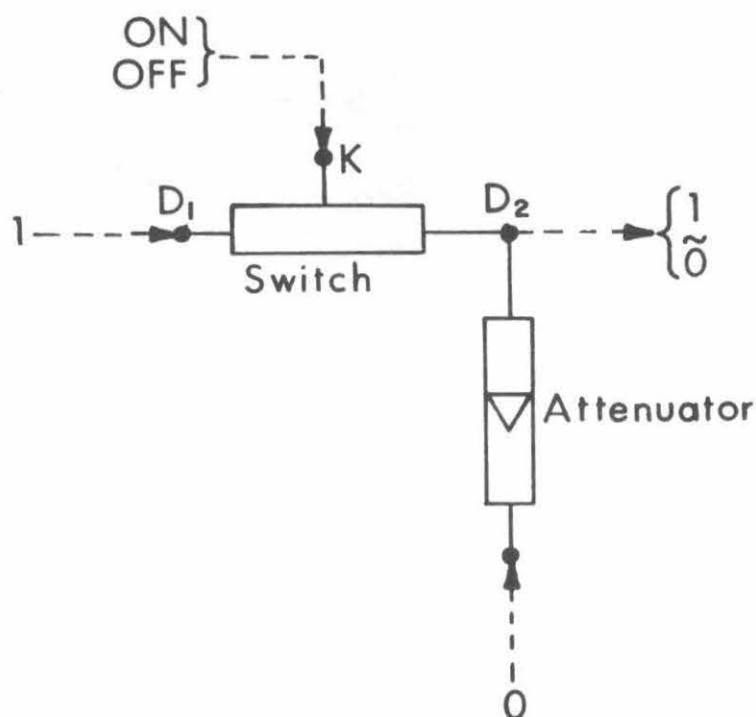


Fig. 4 Typical application of an attenuator.

the behavior of a pullup or pulldown device in an electronic circuit. This may be a resistor or, in the case of VLSI circuits, a load transistor. Since it also converts "strong" to "weak" signals, an attenuator can be regarded as a digital impedance.

The final primitive component we need is an amplifier that converts  $\tilde{0}$  and  $\tilde{1}$  to 0 and 1 respectively. Standard amplifying devices perform this function satisfactorily, hence we denote our amplifiers by the standard triangle symbol of Fig. 5a. Note that an attenuator is the inverse of an amplifier, a fact that guided our choice of symbol for an attenuator. The attenuator symbol contains a reversed amplifier, and also suggests an impedance or load element. In accordance with normal logic design practice we insert a small circle in a line to denote the following nonamplifying inversion operation:

$$1 \rightarrow 0, 0 \rightarrow 1, \tilde{1} \rightarrow \tilde{0}, \tilde{0} \rightarrow \tilde{1}, U \rightarrow U, Z \rightarrow Z.$$

Thus an inverting amplifier can be represented as shown in Fig. 5b.

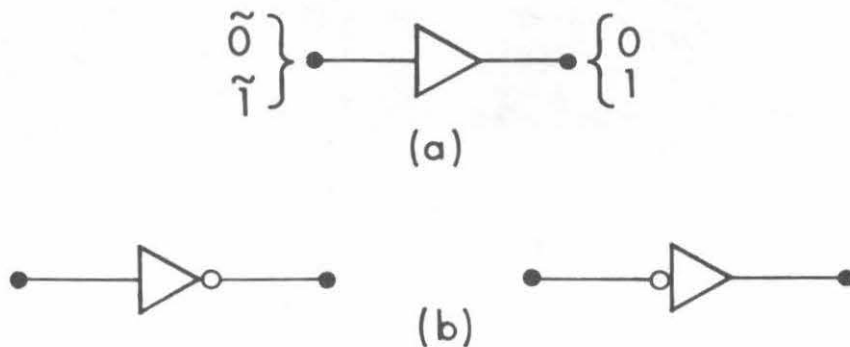


Fig. 5 (a) A non-inverting amplifier. (b) Inverting amplifiers.

### 3. THEORY

We now present a formal description of CSA theory. A CSA network is composed of four basic component types:  $n$ -terminal connectors, three-terminal switches, and two-terminal amplifiers and attenuators. Components are connected via their terminals, where a terminal is the simplest connector. The behavior of a network is determined by the output signal values of its terminals. A set of six logical signal values is recognized:  $V_6 = \{0, 1, \tilde{0}, \tilde{1}, U, Z\}$ . The behavior of all CSA component types is completely defined in terms of  $V_6$ .

#### Connection Function

It is useful to introduce a concept of relative strength among the members of  $V_6$ .

*Definition 1:* Let  $v_1, v_2 \in V_6$ .  $v_1$  is (logically) stronger than  $v_2$ , denoted  $v_1 \cong v_2$ , if  $\#(v_1, v_2) = v_1$  where  $\#$  is defined by Eqns. (1), (2), (5) and (9).

The relation  $\cong$  imposes a partial ordering on  $V_6$  which is depicted graphically in Fig. 6. Clearly  $U$  is stronger than all other member of  $V_6$ , while  $Z$  is weaker than all other values. The values  $0$  and  $1$  are not related by  $\cong$ .

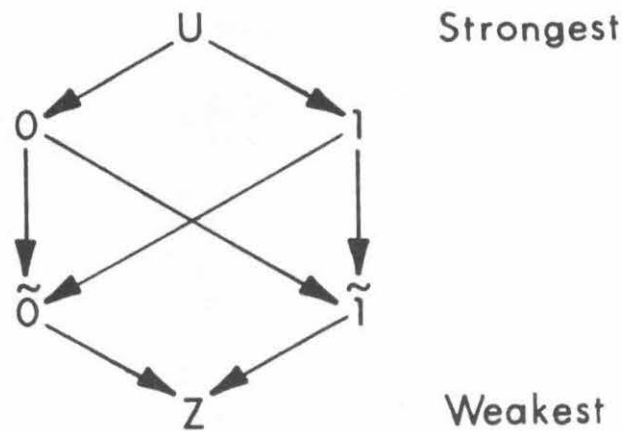


Fig. 6 Relative strength of the logic values in  $V_6$ .

because  $\#(0,1) = U$ ; these values are said to be *contradictory*. Similarly  $\tilde{0}$  and  $\tilde{1}$  are contradictory. Using the foregoing notions, we can now generalize Eqns. (1), (2), (5) and (7) to obtain the following concise definition of the behavior of a connector.

Definition 2: (The  $k$ -place *connection function*  $\#$ ) Let  $C$  be a connector to which the input signals  $v_1, v_2, \dots, v_k \in V_6$  are applied; cf. Fig.2.  $C$  generates a unique output signal  $v(C) = \#(v_1, v_2, \dots, v_k) \in V_6$  defined as follows. If  $v_1, v_2, \dots, v_k$  contain no contradictory values, then

$$\#(v_1, v_2, \dots, v_k) = v_i$$

where  $v_i \cong v_j$  for all  $j = 1, 2, \dots, k$ . If  $v_1, v_2, \dots, v_k$  contain contradictory values, then

$$\#(v_1, v_2, \dots, v_k) = U.$$

The action of  $\#$  on the members of  $V_6$  determines the interpretation of these logical quantities in practical digital circuits.  $Z$  is the logical value of an isolated connector, and also corresponds precisely to the high-impedance state used in tri-state circuits.  $0$  and  $1$  correspond to the usual Boolean variables  $0$  and  $1$ . Here, however, they are seen as strong signals that can override their weaker counterparts  $\tilde{0}$  and  $\tilde{1}$ . Thus  $0$  and  $1$  denote sig-

nals with high drive capability, such as power, ground, and amplifier output signals.  $\tilde{0}$  and  $\tilde{1}$  represent weak signals that have relatively low drive capability; such signals are typically produced by passive load devices. Note that the  $\tilde{0}$  and  $\tilde{1}$  signals are easily mapped onto 0 and 1 respectively by passing them through an amplifier. *Thus from the viewpoint of implementing Boolean functions, we may choose either 0 or  $\tilde{0}$  to represent Boolean zero, and either 1 or  $\tilde{1}$  to represent Boolean one.* U represents a conflict resulting from the simultaneous application of contradictory Boolean values of equal strength to a connector. U is not normally encountered in properly designed or "well-behaved" circuits.

The standard Boolean operations AND and OR which do not require inversions can be implemented by means of a connector alone. Suppose, for example, that  $k$  devices have their output terminals  $z_1, z_2, \dots, z_k$  joined by a connector  $C$ . Let the values  $v_1, v_2, \dots, v_k$  applied to  $C$  via the  $z_i$  terminals be restricted to the subset  $\{\tilde{0}, 1\}$  of  $V_6$ . Then  $v(C)$ , which is defined by Def. 2, implements the OR function. This is because a (strong) 1 applied to any terminal of  $C$  overrides a (weak)  $\tilde{0}$  applied to any other terminal of  $C$ . Similarly, when the  $v_i$ 's are confined to  $\{0, \tilde{1}\}$ ,  $C$  implements the AND function. It is believed that this type of "wired logic" underlies the behavior of all switching circuits, including both branch and gate-type circuits.

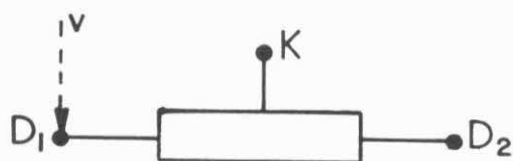
### CSA Networks

A switch, as indicated by Fig. 3, contains a control terminal  $K$  and two symmetric data terminals  $D_1$  and  $D_2$ . The switch is *homogeneous* if the values that all three terminals can assume are identical, that is,  $v(K) = v(D)$ . A manual on-off switch is not homogeneous because  $v(K)$  and  $v(D)$  are defined in incompatible mechanical and electrical domains, respectively. An MOS transistor is homogeneous if the  $K$  input (the gate) and the  $D$  inputs (the source and drain) all employ the same digital voltage levels. An inhomogeneous switch is useful as a transducer between physically incompatible signal domains. Here we will restrict our attention to homogeneous switches where all three terminals may assume values from  $V_6$ .

Figure 7 defines the behavior of the most basic switch in terms of  $V_6$ . The switch is turned on and off by the  $K$  values 1 and 0, respectively; there is

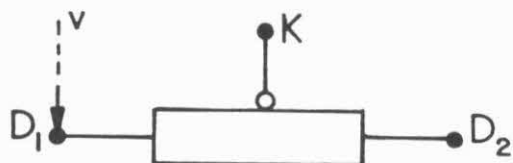


no amplification or gain associated with this switch. It is also convenient to treat the other three switches of Fig. 7 as basic components of CSA networks. In the switch of Fig. 7b the polarity of the K terminal is reversed. The switches of Figs. 7c and 7d can be switched on or off by the weak control signal values  $\tilde{1}$  and  $\tilde{0}$ , thus they have built-in gain. These switches directly model the digital behavior of NMOS and PMOS transistors, respectively. In all cases we make the somewhat arbitrary assumption that  $v(D_1) = v(D_2) = U$  when either Z or U is applied to K. It should be noted that other definitions of switch behavior may be more appropriate for some technologies or circuit configurations. New switch types can be added to those of Fig. 7 as required.



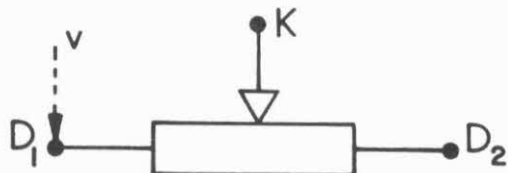
$v(K)$	$v(D_1)$	$v(D_2)$	Switch State
0	v	Z	Off
1	v	v	On
$\tilde{0}, \tilde{1}, U, Z$	U	U	Undefined

(a)



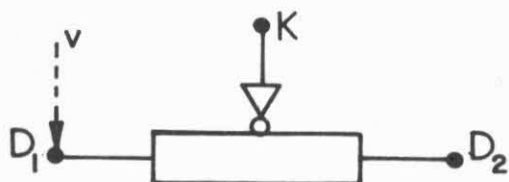
$v(K)$	$v(D_1)$	$v(D_2)$	Switch State
1	v	Z	Off
0	v	v	On
$\tilde{0}, \tilde{1}, U, Z$	U	U	Undefined

(b)



$v(K)$	$v(D_1)$	$v(D_2)$	Switch State
$\tilde{0}, 0$	v	Z	Off
$\tilde{1}, 1$	v	v	On
$U, Z$	U	U	Undefined

(c)



$v(K)$	$v(D_1)$	$v(D_2)$	Switch State
$\tilde{1}, 1$	v	Z	Off
$\tilde{0}, 0$	v	v	On
$U, Z$	U	U	Undefined

(d)

Fig. 7 Four basic switch types and their behavior.

Figure 8 formally defines attenuators and amplifiers. It is easily shown that we can dispense with explicit amplifiers if amplifying switches of the kind appearing in Figs. 7c and 7d are available.

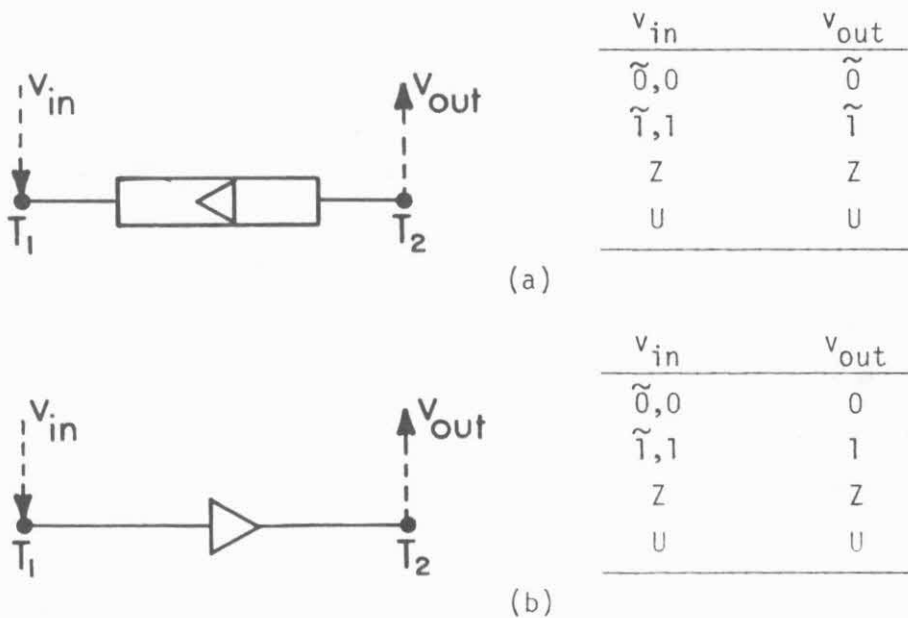


Fig. 8 Definition of (a) an attenuator, (b) an amplifier.

A *CSA network* may be defined as a set of switches, attenuators and amplifiers linked by connectors. The interconnection rules that yield well-behaved networks can be defined in several ways depending on the application at hand. In most cases we require network output signals to be confined to the subset  $\{0, 1, \tilde{0}, \tilde{1}\}$  of  $V_6$ . Z is a tolerable output value also, since an attenuator can be used to convert it to  $\tilde{0}$  or  $\tilde{1}$ . Only U represents an intolerable conflict state that cannot be overridden. We can generalize the concept of a CSA network to include as primitive components or cells, any devices whose behavior can be described as a function on  $V_6$ . This includes all standard combinational and sequential circuits that are defined on the Boolean subset  $\{0, 1\}$  of  $V_6$ . Thus we can easily add high-level primitives like PLAs, decoders, registers, etc. to CSA-type circuits.

Figure 9 shows two simple CSA networks that implement the logical OR operation defined on  $\{0, 1\}$ . That of Fig. 9a uses a parallel configuration of switches which is the standard branch (contact network) implementation of OR;

the attenuator makes the circuit act as a unidirectional OR gate. As this example suggests, both branch-type and gate-type networks can be modeled efficiently by means of CSA networks. However, CSA networks also include many useful structures that are not covered by classical switching theory; the OR network of Fig.9b is an example.

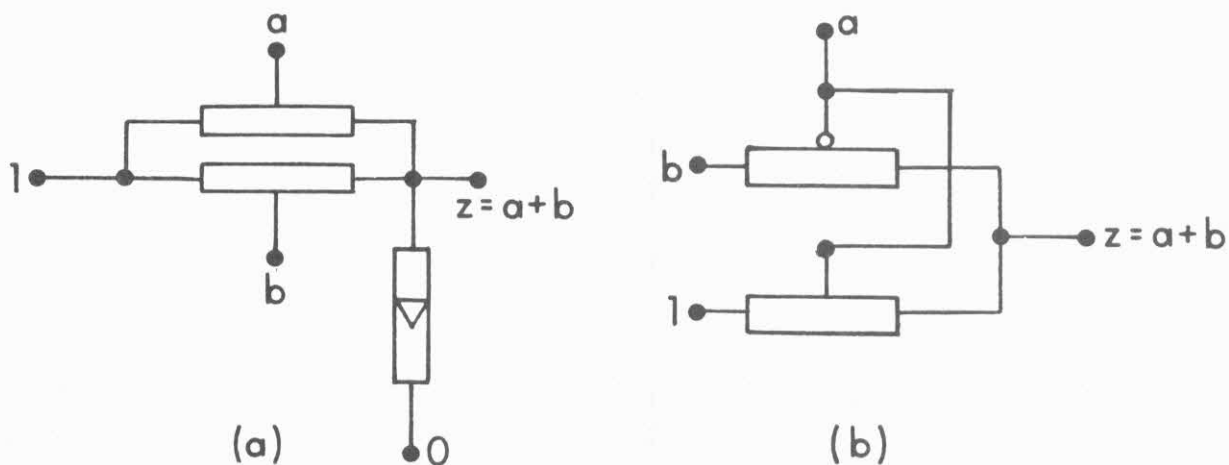


Fig. 9 Two CSA implementations of the OR operation.

### Characteristic Equations

As in standard logical design it is useful to be able to analyze CSA networks by means of algebraic functions and equations. While it is possible to represent CSA network behavior directly in terms of 6-valued functions such as  $\#$ , it seems to be more useful to employ two-valued Boolean functions. To this end we associate a Boolean function  $z^v(C)$  with each connector  $C$  such that  $z^v(C) = 1$  whenever  $v(C) = v \in V_6$ , and  $z^v(C) = 0$  otherwise. The behavior of  $C$  can therefore be fully defined by means of the six functions  $z^0(C)$ ,  $z^1(C)$ ,  $z^{\bar{0}}(C)$ ,  $z^{\bar{1}}(C)$ ,  $z^Z(C)$ ,  $z^U(C)$ , which we term the *characteristic (Boolean) functions* of  $C$ . Characteristic functions for the basic CSA elements are easily derived. They can be described by truth tables as in Figs. 7 and 8, or by means of Boolean equations. Boolean equations that define the characteristic functions of the output terminals of a CSA element are called its *characteristic equations*. By combining the characteristic equations of the individual components, characteristic equations describing the logical behavior of any CSA network can be constructed.

If  $a$  and  $b$  can assume any values from  $V_6$ , the characteristic equations have the following slightly more complex forms:

$$z^1 = a^1 b^1 + a^1 b^0 + a^0 b^1$$

$$z^{\tilde{0}} = a^0 b^0$$

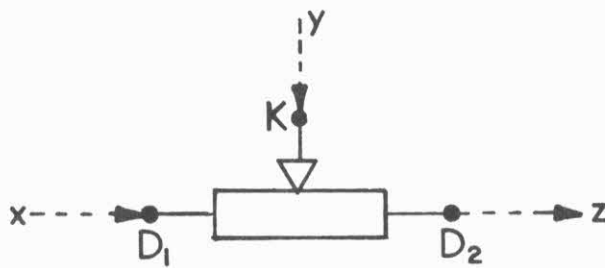
$$z^U = a^{\tilde{0}} + a^{\tilde{1}} + a^U + a^Z + b^{\tilde{0}} + b^{\tilde{1}} + b^U + b^Z$$

$$z^0 = z^{\tilde{1}} = z^Z = 0.$$

#### 4. APPLICATIONS

A major objective of introducing CSA networks is to obtain logically consistent replacements of the mixed-type diagrams widely used in VLSI design specification. Figure 11a shows a representative mixed circuit  $N_1$  (from plate 4 in [1]) which contains three distinct and basically incompatible types of component symbols: logic symbols (the inverter  $I$ ), standard electronic symbols (the pullup transistor  $T_0$ ), and "stick" IC layout symbols (various connectors and the transistors  $T_1$ ,  $T_2$  and  $T_3$ ). Figure 11b shows the CSA equivalent circuit  $N_2$  whose behavior can be rigorously defined in terms of  $V_6$  without appealing to electronic or IC layout concepts. Note that there is a one-to-one correspondence between the components and connectors of  $N_1$  and those of  $N_2$ . Thus the CSA network  $N_2$  can be used in the same manner as  $N_1$  to provide an approximate indication of the geometry of an IC implementing these circuits. Hence CSA circuits appear to be useful in the preliminary description of IC layouts. If desired, the connector (color-) coding scheme of Fig. 11a can also be used in CSA networks.

We next demonstrate the utility of CSA theory in a small but important aspect of VLSI design, the synthesis of inverters from MOS transistors. Classical logic design reveals none of the internal structure of inverters, which significantly affects their layout cost. Inverters are thoroughly analyzed from a VLSI viewpoint in [1,5] using electronic circuit models. We give here a parallel analysis for certain types of inverters using CSA logical models.



$$z^0 = x^0 y^T$$

$$z^1 = x^1 y^T$$

$$z^{\tilde{0}} = x^{\tilde{0}} y^T$$

$$z^{\tilde{1}} = x^{\tilde{1}} y^T$$

$$z^Z = y^F + x^Z y^T$$

$$z^U = y^Z + y^U + x^U y^T$$

Fig. 10 Definition of switch behavior by means of characteristic Boolean equations.

Figure 10 shows a set of characteristic equations for the switch of Fig. 7c. It is convenient in such equations to let T (true) denote either 1 or  $\tilde{1}$ , and F (false) denote either 0 or  $\tilde{0}$ . Similar equations defining connectors, attenuators, amplifiers, etc. can readily be obtained. Surprisingly, the characteristic equations for a connector (which must be equivalent to Def. 2 above) are relatively complex. For instance, the conditions under which the connector output value is  $\tilde{1}$  can be specified as follows.

$$z^{\tilde{1}} = (v_1^{\tilde{1}} + v_2^{\tilde{1}} + \dots + v_k^{\tilde{1}})(v_1^{\tilde{1}} + v_1^Z)(v_2^{\tilde{1}} + v_2^Z) \dots (v_k^{\tilde{1}} + v_k^Z).$$

Despite this apparent complexity, the characteristic equations for well-behaved logic networks can often be reduced to forms that closely resemble standard Boolean equations. Consider, for instance, the OR network of Fig. 9a. Assume that the values of the inputs a and b are confined to the subset  $\{0,1\}$  of  $V_6$ , which will be the case in any well-behaved CSA network employing this particular OR gate. Then the output function z always assumes values from  $\{\tilde{0},1\}$  according to the following set of characteristic equations:

$$z^1 = a^1 + b^1$$

$$z^{\tilde{0}} = a^0 \cdot b^0$$

$$z^0 = z^{\tilde{1}} = z^U = z^Z = 0.$$

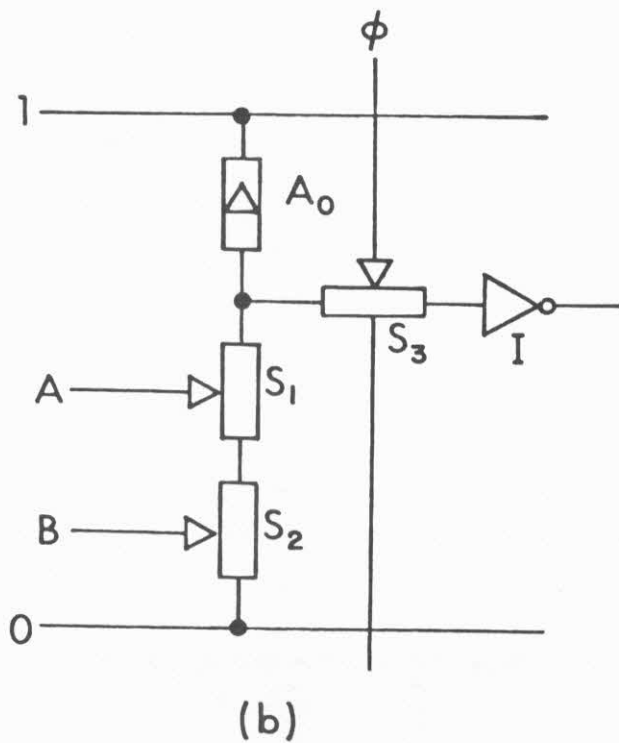
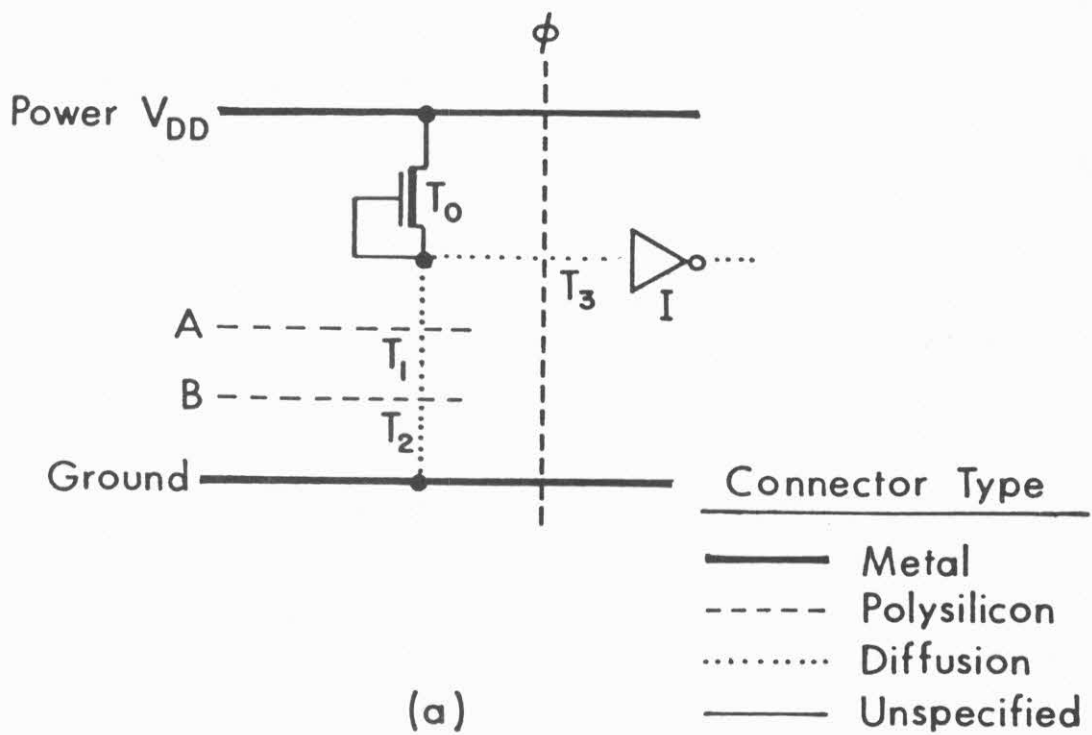


Fig. 11 (a) A typical mixed circuit. (b) The equivalent CSA network.

The building blocks of most VLSI circuits are MOS transistors that are configured either as transistor switches or a pullup/pulldown loads; of Fig. 11. We therefore need only two CSA component types, switches and attenuators. The switches may be amplifying or nonamplifying, inverting or non-inverting, depending on the particular MOS technology being used. It is easily proven that at least two CSA components are needed to build an inverter, and that the simplest inverter must contain either two switches, or a switch and an attenuator. Figure 12a shows an inverter composed of two switches of opposite K polarity. The fact that it inverts  $\{0,1\}$  can be quickly proven as follows. Suppose that the primary input  $x$  is 0. This causes switch  $S_1$  to close making  $a=1$ , while simultaneously  $S_2$  opens making  $b=Z$ . The primary output  $z$  therefore assumes the value  $v(T_2) = \#(a,b) = \#(1,Z) = 1 = \bar{x}$ . Conversely if  $x=1$ , then  $a=Z$ ,  $b=0$ , and  $z = \#(Z,0) = 0$ . The behavior of this network can also be defined by the following characteristic equations:

$$z^0 = x^1$$

$$z^1 = x^0$$

$$z^{\tilde{0}} = z^{\tilde{1}} = z^U = z^Z = 0.$$

Hence for any  $x \in \{0,1\}$ ,  $z = \bar{x}$ , so the network represents a Boolean NOT gate or inverter. If the switches are replaced by amplifying versions (see Fig. 7), then the network of Fig. 12a also performs the inversions  $\tilde{0} \rightarrow 1$  and  $\tilde{1} \rightarrow 0$ . It models very closely both the structure and logical behavior of the CMOS inverter [5] appearing in Fig. 12b.

Figure 12c shows another CSA inverter composed of two elements, this time a switch and an attenuator. If the input signal  $x$  is  $\tilde{0}$  or 0, then  $S_1$  opens making  $a=Z$ . The attenuator's output signal  $b$  is  $\tilde{1}$ , therefore the primary output value  $z$  is determined by  $\#(a,b) = \#(Z,\tilde{1}) = \tilde{1}$ . Similarly if  $x = \tilde{1}$  or 1, then  $a=0$ , and  $z = \#(0,\tilde{1}) = 0$ . Notice that in one case  $z$  assumes the weak 1-like value  $\tilde{1}$ . This asymmetric lack of drive is indeed a basic characteristic of the NMOS ratio-type inverter of Fig. 12d [1] which is represented at the CSA level by the network of Fig. 12c. Thus the CSA model describes the

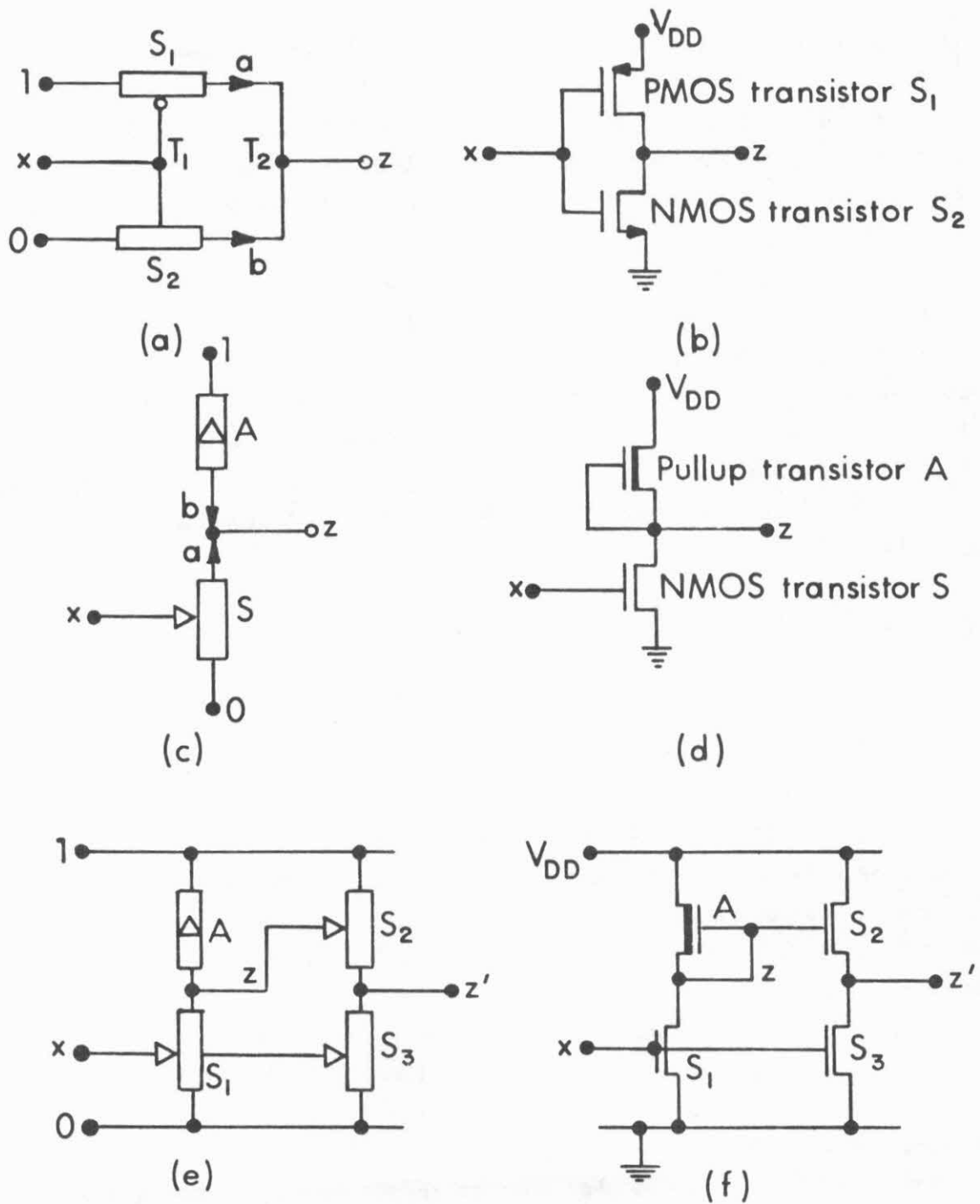


Fig. 12. Various CSA inverter designs and the MOS circuits they model.



structure, logical behavior, and drive capability of this NMOS inverter.

We can attempt to amplify the weak  $\tilde{1}$  output signal of the CSA network in Fig. 12c by adding more components. Figure 12e shows the most straightforward way of doing this. A second switch  $S_2$  is inserted in the line  $z$  to convert the offending  $\tilde{1}$  to 1; it does so simply by connecting the "power line" to the new primary output  $z'$ . To ensure that  $z'$  is also driven to 0 when  $x$  is  $\tilde{1}$  or 1, we need the third switch  $S_3$ .  $S_3$  is controlled by  $x$  and connects the "ground line" to  $z'$ . The result is an inverter that always generates the strong output values 0 and 1. As Fig. 12f indicates, we have reinvented the inverting super buffer [1], a standard circuit in VLSI design. It is used to replace the ratio-type inverter of Fig. 12d in situations where the asymmetric output values 0 and  $\tilde{1}$  of the latter are unacceptable.

## 5. DISCUSSION

The logical design theory presented here can be regarded as a refinement of standard two-valued switching theory which reveals some hidden variables and components that are of central importance in VLSI design. CSA components have the same number of terminals as the MOS devices they model, so that a CSA network displays all the connectors of a circuit. Explicit representation of these connectors is essential in IC chip layout. Our approach also emphasizes the important role played by connectors in logical operations. CSA networks use attenuators to model pullup/pulldown devices which are hidden in classical logical design. These devices are at least as important as switches for example, they often occupy more chip area than simple switches. The new variables  $\tilde{0}$  and  $\tilde{1}$  allow a simplified concept of drive capability to be applied to logic circuits, while the variable  $Z$  represents a rigorous logical definition of the well-known high-impedance state.

CSA networks can also be viewed as simplified electronic circuits in which only a few limiting values of the analogue signals being processed are retained. These are the logic values of interest in digital design, and are represented by  $V_6$ . A CSA switch thus models only the digital behavior of a transistor switch, while an attenuator is a kind of digital resistor. The voltage-current signal values of a resistor  $R$  can vary over a continuous range; those of an attenuator  $A$  are confined to  $V_6$ . Unlike  $R$ ,  $A$  has only two

modes of operation. In the first mode A is "non-conducting" where the output values of its terminals are both F ( $\tilde{0}$  or 0), or both T ( $\tilde{1}$  or 1). Thus there is no "logical potential" across A. In the other "conducting" mode of operation, one terminal of A has the value T, while the other has the value F.

CSA theory provides a uniform and rigorous methodology for VLSI design. It employs logical component types that accurately model the digital behavior and external interconnection requirements of MOS transistors. (Bipolar transistors can also be modeled.) It thereby eliminates much of the need for mixed logic/electronic models, and reduces VLSI design at this level to a logical rather than an electronics design process.

Much work remains to be done to determine the extent to which the CSA model can be used by VLSI designers. Of particular interest is the development of CSA-based design algorithms. These algorithms can be expected to be quite rigorous, and should be easy to incorporate into computer-aided design systems. CSA theory also appears to be promising for the analysis of the failure modes and testing requirements of VLSI systems. Unlike standard logic diagrams, a CSA network displays essentially all the connectors of a circuit, hence the standard connector stuck-at-0/1 fault model [4] can be used more accurately. Furthermore this fault model can be augmented by stuck-at- $\tilde{0}/\tilde{1}/Z$  fault types. Stuck-at-Z faults are known to occur frequently in practice [3]. Stuck-at- $\tilde{0}/\tilde{1}$  faults can model certain types of load-dependent failures.

#### REFERENCES

- [1] C. Mead and L. Conway: *Introduction to VLSI Systems*, Addison-Wesley, Reading, Massachusetts, 1980.
- [2] M. A. Harrison: *Introduction to Switching and Automata Theory*, McGraw-Hill, New York, 1965.
- [3] R. L. Wadsack: "Fault modeling and logic simulation of CMOS and MOS integrated circuits," *Bell System Technical Journal*, vol. 57, pp. 1449-1474, May-June 1978.
- [4] M. A. Breuer and A. D. Friedman: *Diagnosis and Reliable Design of Digital Systems*, Computer Science Press, Woodland Hills, California, 1976.
- [5] W. N. Carr and J. P. Mize: *MOS/LSI Design and Application*, McGraw-Hill, New York, 1972.

ARCHITECTURE SESSION

*Chairperson: ALAN L. DAVIS*  
*Associate Professor of Computer Science*  
*University of Utah*

## ARCHITECTURE SESSION

With the dawn of the VLSI era, it is clear that the conceptual framework encompassing the design of a chip must expand considerably. In particular, the view of a chip as a component must change to the view that a chip is either a system, or is a part of a system.

One view that has been taken previously is that as the semiconductor technology allows the fabrication of chips which contain larger quantities of transistors, what should be done is to build existing systems on a single IC or on a small set of IC's. Examples of this approach can be seen in any of the microprocessors which are available today. Today's microprocessors are exciting primarily due to the fact that they are small, consume very little power, and are very cheap. What is not interesting, and in fact is almost a tragedy, is that the architecture and programming methodology of these microprocessors is so reminiscent of the computers the late 1950's. This view may be fine for making better washing machines, typewriters, cars, watches, and slide rules. However, this view will certainly limit our ability to design and use the class of computers which the fabrication technology allows.

Today's designers are using LSI components to build novel architectures which represent significant improvements in cost and performance over previous systems. One notable characteristic of these systems is that while the LSI chips contain the majority of the system's transistors, the majority of the system's chips and pins are tied up in "glue chips." Since the system cost is approximately linear with the number of pins, the major cost of such systems results from the "glue" and not from the components which provide the bulk of the system's function. This observation indicates that the present approach is not good enough.

The VLSI era also presents a tremendous challenge to the architects of the 80's in that they must understand some of the architectural restrictions within the chip. It is certain that we do not clearly understand all of the implications of the new technology, but a few issues are emerging as clear candidates for consideration. It is clear that communication constraints on a VLSI system are more stringent than those on systems where discrete wires are used to interconnect the system components.

Signals on a VLSI chip are carried over a relatively few layers of interconnect. Communication bandwidth is effectively limited by the number of wires that can be routed through a given area, by the energy and driver area required to switch the voltage on wires, and by the diffusion delay within the wire itself. This communication problem becomes more tractable in structures with exclusively local communication, a feat that can be accomplished most easily in a highly regular arrangement of functional elements.

A related VLSI constraint is that the pins to the "off-chip world" are a scarce resource. These pins represent enormously slow communication ports due to the relatively huge capacitive loads they present to the

drive circuitry that must be built on the chip. Such issues must be taken into account in future VLSI designs.

There are several directions that seem important and valid for future VLSI architectures. The papers in this session focus on some of these topics.

As design tools and the "silicon foundrys" mature, designers will be able to design chips to solve computational problems directly on silicon, and will not have to rely on microprocessors and lots of software to implement their designs. Specialization of function, particularly of the communication arrangement in the case of VLSI, is an important and powerful architectural technique. The paper by Liu is an example of this approach.

Another direction is the organization of chip designs which significantly reduce the amount of "glue" circuitry necessary to create highly concurrent systems. The Smith paper describes how minor changes in conventional microprocessor architectures can ease the glue requirements by providing more flexible communication structures.

The Budzinski paper presents a more aggressive effort in the description of a single chip system which is capable of reconfiguration to support the function of a single wide word processing engine or a collection of smaller word processors capable of supporting vertical and/or horizontal concurrency.

The two papers by Martin, and by Browning and Seitz address the more fundamental issue of communication in VLSI systems which derive their power from ultra-high levels of concurrency within a highly regular replication of processing elements. This is an approach which is highly distinctive to VLSI, and is both driven by and made possible by this new technology.



A RESTRUCTURABLE INTEGRATED CIRCUIT  
FOR IMPLEMENTING PROGRAMMABLE DIGITAL SYSTEMS

ROB BUDZINSKI  
JOHN LINN  
SATISH THATTE

TEXAS INSTRUMENTS INCORPORATED  
DALLAS, TEXAS

Copyright -C- 1981 Texas Instruments Incorporated

This work is funded in part by the Defense Advanced Research Projects Agency  
under contract No. MDA 903-79-C-0433.

## ABSTRACT

The Restructurable Integrated Circuit, a highly flexible and programmable multimicrocomputer is presented. The goal of this integrated circuit is to apply the large number of gates that are available on a custom designed VLSI IC to the design of a highly flexible integrated circuit. The main application of the Restructurable Integrated Circuit (RIC) will be the implementation of digital system hardware through programming of a RIC. The flexibility provided within the RIC includes: user definable micro language, user programmable assembly language, user programmable microcode, dynamic coordination of multiple internal processors, coordination of processors on multiple RICs, internal memory use as caches or as a member of a virtual memory hierarchy, general topology for interchip communication and external data paths, and a user definable interrupt mechanism. By providing this high degree of flexibility, the cost and reliability advantages of high volume production can be accrued, while providing performance comparable to a custom VLSI IC.

## 1. INTRODUCTION

The concept of a highly programmable, restructurable VLSI integrated circuit is an extremely important step towards achieving the maximum impact from VLSI. Not only do programmability and flexibility provide new creative opportunities for the system designer, but they help overcome two major obstacles to pervasive use of VLSI, as well.

Design cost and design cycle time are critical barriers to implementing VLSI systems. Typically a state-of-the-art LSI custom design consisting of 5K gates and 5K bits of read-only memory costs approximately \$500K and takes about 18 months to design and layout. This works out to about \$100 per gate in the design. Even if, over the next five years, design costs are reduced by an order of magnitude to \$10 per gate, a typical state-of-the-art VLSI custom design in this time frame, consisting of 50K gates and 50K bits of read-only memory, will still cost \$500K to design. A very flexible restructurable VLSI chip that can be structured to provide a wide range of capabilities with state-of-the-art performance presents a viable alternative to custom designs in low volume applications.

Reliability, testing and maintenance considerations are extremely important in complex VLSI systems. When reflected in the cost of service calls or returned products, poor reliability can contribute more to system lifetime cost than the initial manufacturing cost. Traditionally, reliability has improved with each increase in the level of integration. However, reliability benefits from accumulated learning, as shown in Figure 1. A generic programmable chip, in this case the Texas Instruments TMS-1000 microcomputer, increases in



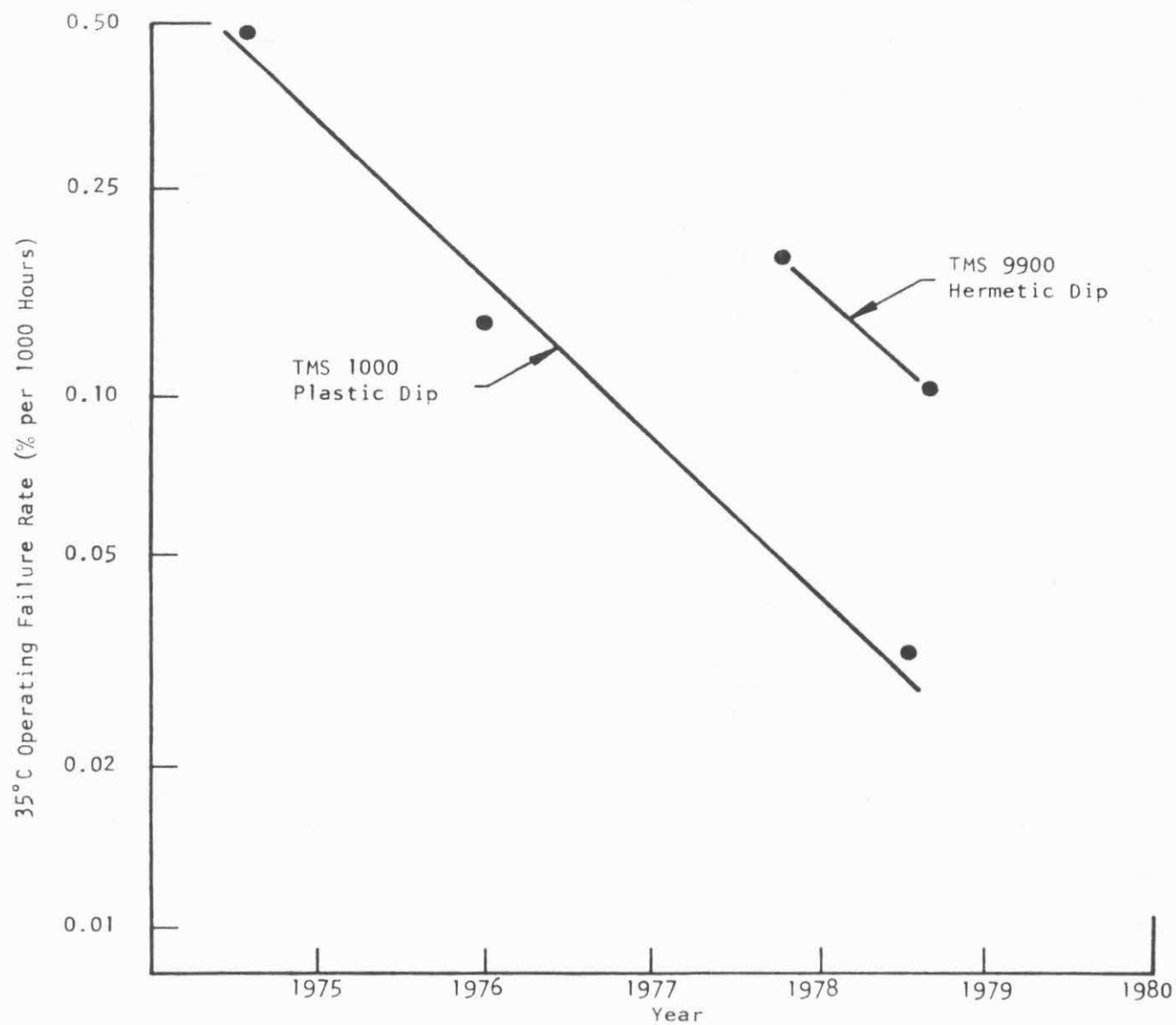


FIGURE 1. Microprocessor Failure Rate History

reliability as more are manufactured. Although any specific programming of the chip may not be made in any significant volume, each programming still benefits from the reliability improvements of the generic device. Custom designs, even using an identical process, do not benefit significantly from the high volume reliability learning of other chips. A restructurable VLSI circuit will provide a high degree of reliability learning, even though the volume of most programming types may be small.

The RIC is a semi-custom IC similar in purpose to gate arrays or master-slices. The gate array approach allows a logic diagram to be translated into silicon through software. The software creates an interconnect pattern between the gates so that the logic diagram is implemented in silicon. The gate array approach is very flexible, but the gate array approach does not provide any special structure for implementing programmable digital systems. The RIC approach uses the large number of gates on a VLSI IC to build a chip which is highly flexible for implementing programmable digital systems. The RIC approach differs from gate arrays in that the RIC has the vast majority of silicon committed to a specific design. The flexibility of the RIC is achieved through the design of a programmable mechanism for controlling the hardware resources on the chip. The block diagram of the Restructurable IC is shown in figure 2.

The RIC is a multimicrocomputer containing four 16 bit processors called Microprogrammable Slices (MPSs). The MPS resources can be controlled at two basic levels. One level is the coordination of MPSs. The four MPSs can be dynamically configured at run time into any combination of three fundamental structures. One structure is the lockstep in which two or more MPSs are structured to form a wider word computer. This structure is formed by directing the same microinstruction stream to all of the MPSs in the lockstep and structuring the arithmetic status, carry chain, and shift/rotate linkage to configure the MPSs into a wider word computer. The second fundamental structure is independent MPSs. In this structure each MPS has its own microinstruction stream. A set of array processors can be structured in the independent configuration by directing the same microinstruction stream to the MPSs without any coordination of arithmetic signals between MPSs. The third fundamental structure is pipelined MPSs. Each MPS forms a stage in the pipeline. The microinstruction streams are different for each stage. An internal data bus within the RIC provides for simultaneous sending and receiving of data between adjacent stages (MPSs) in the pipeline.

The other basic level for controlling MPSs is language interpretation. The language interpretation structure is programmable at two levels. One level is a definable vertical microcode and/or assembly code. The other level of programming is a PLA which interprets the vertical microcode through finite state machines. The vertical microcode and/or assembly code are defined through the contents of the PLA. Microcode can be contained in the on chip ROM or in RAM. MPSs become user microprogrammable by allowing microcode to be contained in the RAM.

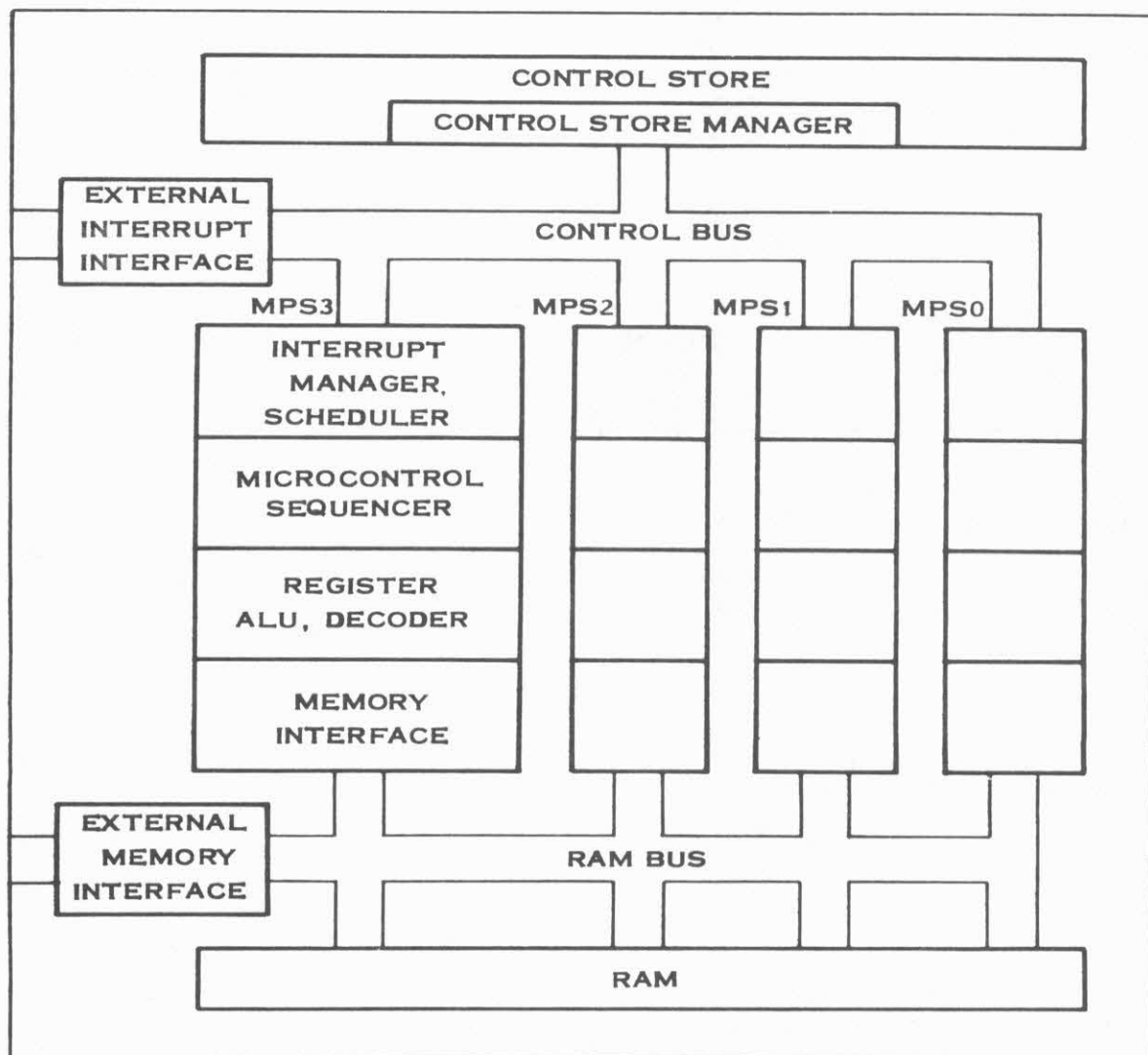


FIGURE 2. Restructurable IC Block Diagram

The RIC also provides for coordination of MPSs contained across multiple RICs. The concept of external coordination of microprocessors has been developed in (1,2). The coordination of MPSs on multiple chips is accomplished with status ports. The status port contains four signals for ALU status, a carry chain, a shift/rotate linkage and a synchronization signal. The external status port interconnection to internal MPS structures is programmable. The use of the RIC status ports provide for implementing computers with word widths greater than 64 bits. Also the status ports provide for flexibility in implementing pipelines with multiple RICs. The external interface of the RIC provides the capability for coordinating independent processors implemented on multiple RICs.

The RIC external interface is composed of three sections: a data port section, a status port section and an interrupt port section. There are two 16 bit wide data ports. The 16 lines are bidirectional and carry addresses and data. Each status port has a general purpose arbitration mechanism. One of three arbitration modes can be used: round robin, master-slave, or a general arbitration method implemented by external hardware. These arbitration methods allow a great amount of flexibility in the communication between RICs, memories, and I/O devices. The interrupt port on the RIC uses the same arbitration methods as the data port. Thus the interrupt port has great flexibility in the topology of the interrupt network. The interrupt port has flexibility in determining the information protocol. The interrupt port can be used for a range of applications from conventional receive only vectored interrupts up to a user defined interrupt driven interchip communication.

The internal memory system of the RIC supports the internal structures of MPSs. There are four memory modules. The memory modules can be accessed in parallel when each MPS accesses its own memory module. Also, the internal RAM bus allows the sharing of the memory modules among the MPSs. The internal RAM bus also can be structured to support pipelined MPSs. In the pipeline structure, each MPS can send and receive data simultaneously. The internal memory system has a capability for memory mapping. Memory mapping allows the internal RAM to be used as a cache or a member of a virtual memory hierarchy.

In the following section, the restructuring capability for the internal coordination of the multiple processors contained in a RIC is discussed. This will be followed by a discussion of configurations of multiple RICs. Then the processing element in the RIC will be described followed by a description of the internal RAM system and the external interface.

## 2. SINGLE-CHIP CONFIGURATIONS

The MPSs of a single RIC can be configured into several modes. The three basic MPS structures are: independent processors, lockstepped processors and

pipelined processors. The configuration used is determined by directing the microinstruction stream. In the independent and pipelined configurations, each MPS receives its own microinstruction stream. In the lockstep configuration, all MPSs in the lockstep receive the same instruction stream. The various MPS structures are discussed below.

In the independent mode there are up to four independent instruction streams on a RIC chip. They operate on four different data streams. The data streams can be completely independent, or they can communicate with one another by passing messages through memory (on-chip or off-chip RAM). Figure 3 illustrates this configuration.

The internal lockstep mode uses a single micro instruction stream to control multiple MPSs. This mode of operation allows a wide word (as wide as 64 bits when all MPSs have the same instruction stream) machine to be designed. Operation in this mode is similar to today's bit-sliced microcomputers (3,4) and it is shown in Figure 4.

The pipeline mode uses multiple instruction streams and multiple data streams. Each MPS implements one stage of the pipeline. The data normally flows unidirectionally between neighboring MPSs. For example, as shown in Figure 5, MPS 3 can be programmed to prefetch the machine instructions from the off-chip main memory, MPS 2 decodes the machine instruction, MPS 1 is programmed to perform address computation and fetches operands from the main memory, and MPS 0 is programmed to do computation specified by machine instructions.

In addition to the structures discussed above, various combinations of these configurations are possible within a single RIC chip. For example, MPS 0 and 1 form one internal lockstep, and MPSs 2 and 3 form another lockstep. The lockstep of MPSs 2 and 3 can emulate the Central Processing Unit (CPU) of a 32 bit machine, while the lockstep of MPSs 0 and 1 can be programmed as a graphics processor, making the system suitable for a high bandwidth graphics application.

### 3. Multi-chip Structures

Multi-chip configurations are used to achieve improved functionality and performance beyond that which is possible with a single-chip configuration. For example, a multi-chip structure may employ one or more RICs as the CPU of a machine, another RIC as an I/O processor, and another RIC as a floating point processor. In this section various multi-chip structures are described.

An external lockstep connects two or more MPSs, each on a different RIC, to

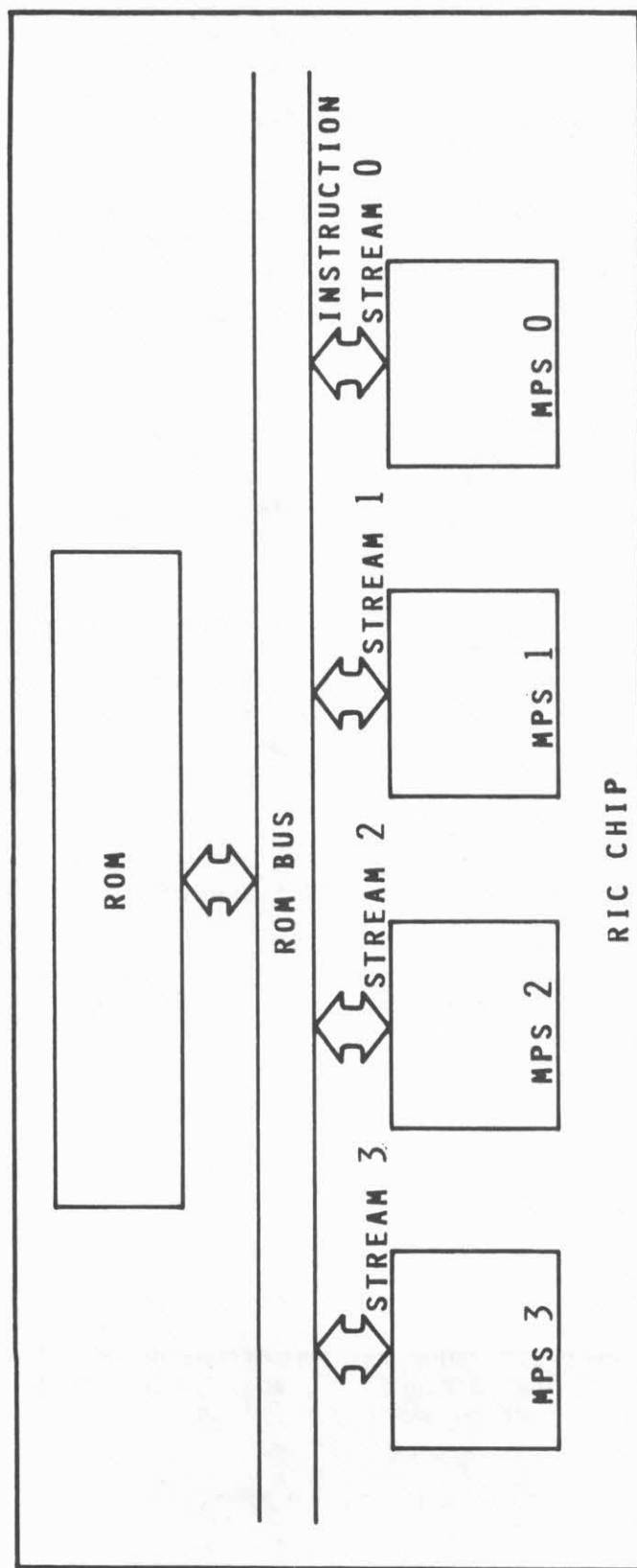


FIGURE 3 RIC INDEPENDENT MODE

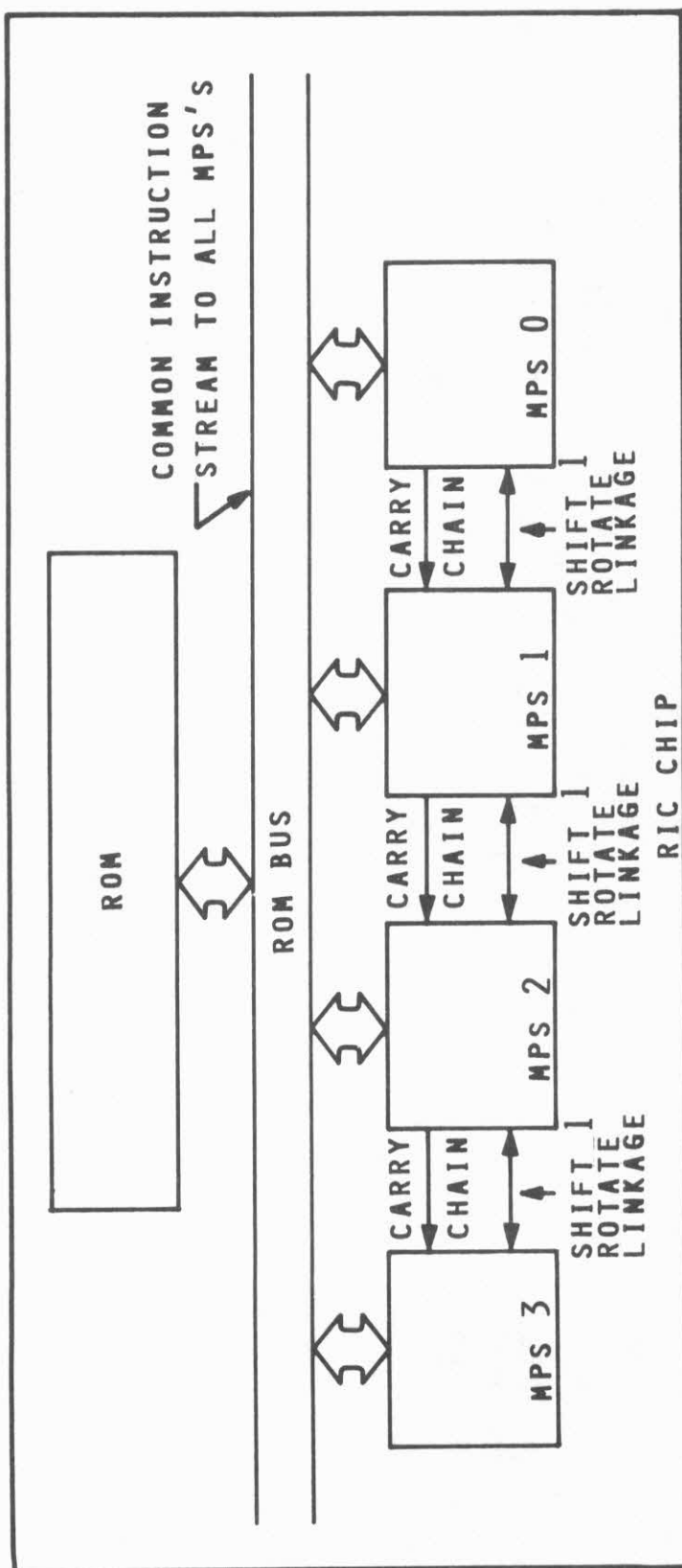


FIGURE 4 RIC INTERNAL LOCKSTEP MODE

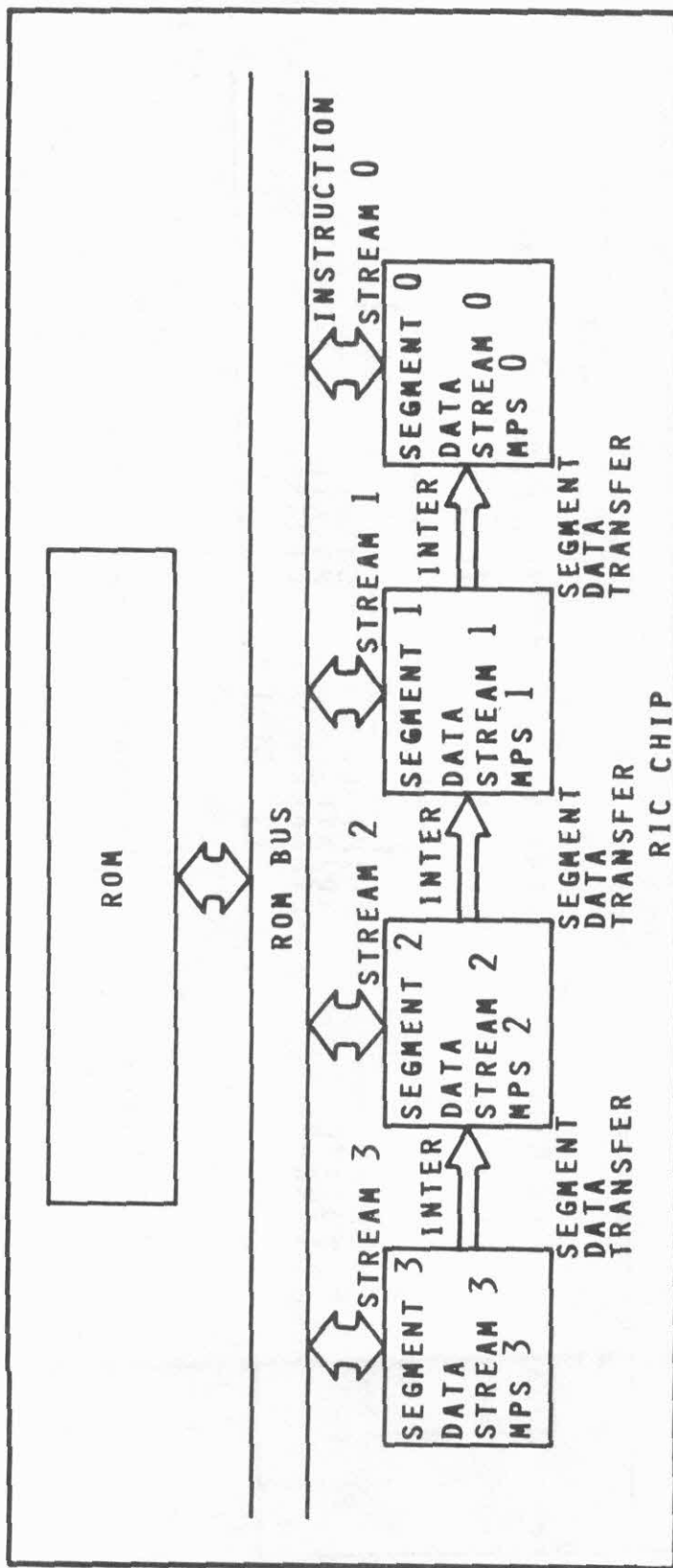


FIGURE 5 RIC PIPELINE MODE



form a lockstep. An external lockstep is illustrated in Figure 6. On each cycle, each MPS in the lockstep executes the same microinstruction, but operates on its own data stream. Since each MPS resides on a different chip each MPS has to fetch its own microinstructions. The status connection synchronizes externally lockstepped MPSs so that they are executing instructions in unison. The status connection also contains ALU result status, a carry linkage, and a shift and rotate linkage.

The hybrid lockstep structure is a lockstep in which MPSs are lockstepped together within one RIC as well as lockstepped externally to MPSs on one or more other RICs. An example of a hybrid lockstep is given in Figure 7.

Two basic types of pipelines can be made with multichip structures. One type, the internal lockstep pipeline has each stage of the pipe formed with an internal lockstep of MPSs. This mode can be used for pipeline widths of up to 64 bits. If the pipe is required to be more than 64 bits wide, each stage of the pipe is formed with a hybrid lockstep. The second type of pipeline structure, the external lockstep pipeline, forms each stage of the pipeline with an external lockstep of MPSs.

In addition to the above structures, the RIC is designed so that combinations of the various internal and external configurations can be combined among various RICs.

#### 4. MicroProgrammable Slice Design

The MPS is the processing element of the RIC. Each MPS contains six major blocks: the data path (computation hardware), the PLA for interpreting instructions for controlling the data path, the ROM address sequencer, the interrupt manager, the scheduler, and the programmable interconnect.

The data path in a MPS is 16 bits wide. It contains a dual port register file of sixteen 16-bit wide registers. The data path contains a high performance ALU. Two registers in the register file can be accessed from two 16 bit wide buses simultaneously. The data path also has a hardware unit to Shift, Extract, and Rotate data called the SERU. In addition to the usual shift and rotate operations in the data path, the SERU can be used to extract fields of a machine instruction being emulated and pass the extracted fields as parameters to the PLA generating control signals for the data path. This PLA also generates signals to coordinate the ROM sequencer. The ROM sequencer is used to generate addresses for the ROM. The sequencer provides for loop control, subroutine calls, branches, and repeating the execution of an instruction.

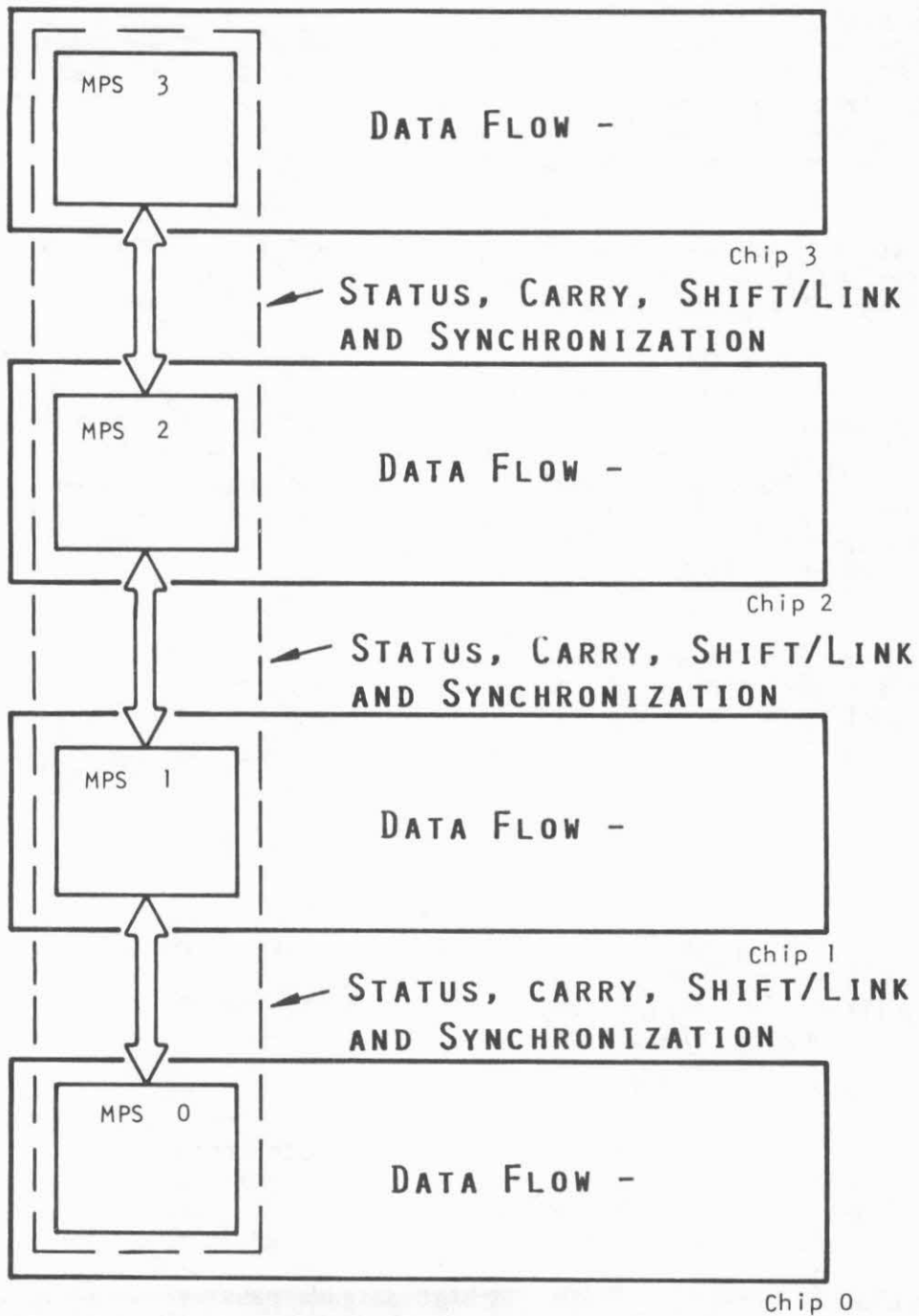


FIGURE 6. External Lockstep

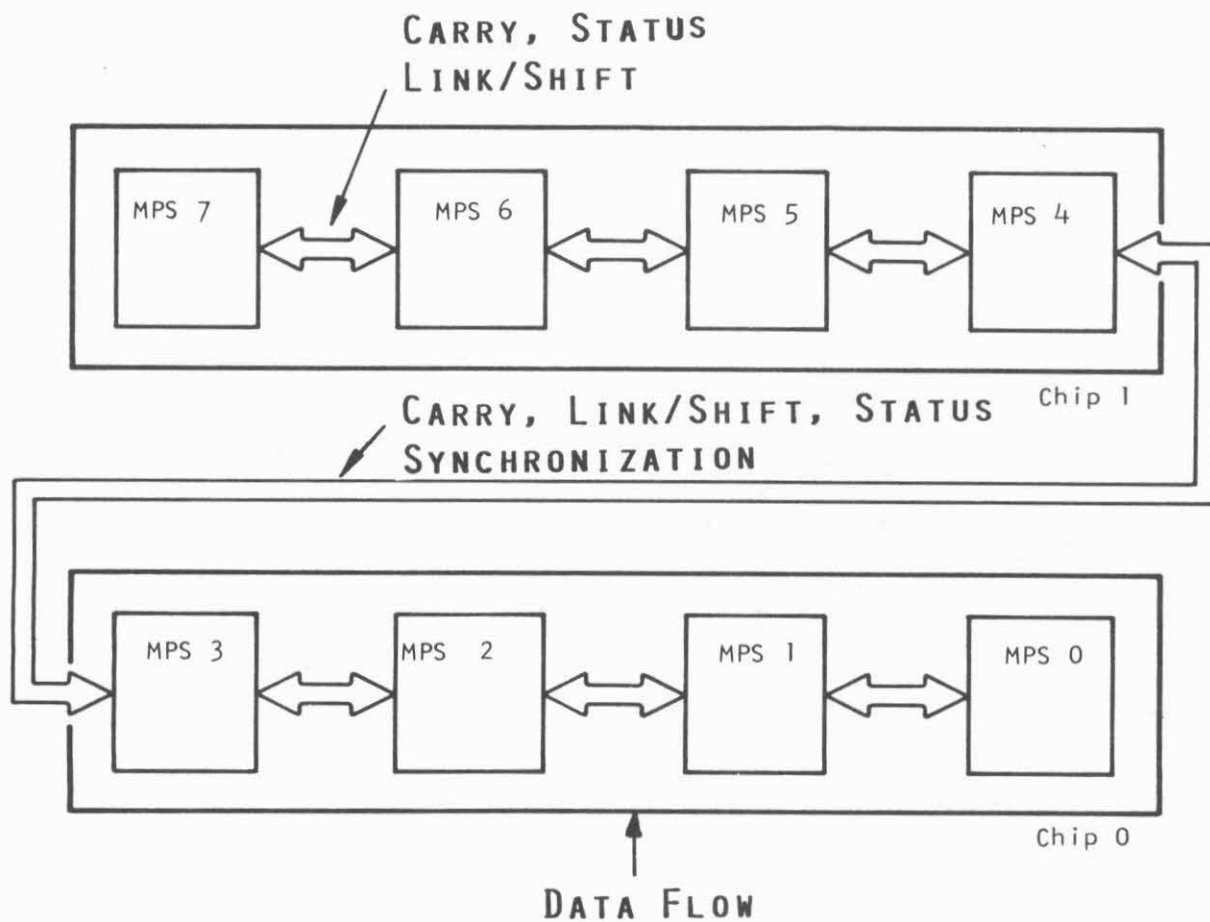


FIGURE 7. Hybrid Lockstep Forming a 128 Bit Lockstep

The MPS is made restructurable to an architecture through four techniques:

- 1 ROM programming
- 2 PLA code stored in each MPS
- 3 Interrupts at the microcode level
- 4 Programmable interconnect

The MPS is designed to interpret vertical microcode or machine code (assembly code) using the PLA. Microcode and low complexity machine code instructions are interpreted directly through the PLA. More complex machine code instructions are interpreted in terms of microinstructions or microroutines. The instructions that are interpreted by an MPS can be contained either in the on-chip ROM, in the on-chip RAM or in an external RAM. This feature provides for user microprograms to be contained in either ROM or RAM. The ROM can contain system microprograms for a variety of tasks: control programs for interrupts, internal memory management, self testing, initiating internal MPS structures, and initiating external RIC structures. The ROM can also contain microprograms for interpreting machine languages. A RIC can interpret multiple microcode languages and/or multiple machine languages. A PLA within an MPS is programmed to interpret a particular language. This PLA can be programmed to interpret more than one language, depending upon the languages. Since there are four MPSs on a RIC, at least four different languages could be interpreted. It is expected that a RIC will interpret existing languages (emulation) as well as interpret languages created for a particular application. For example, a language could be created for: instruction prefetch, instruction decode, address calculation, self testing, memory management, or any other computation task.

The information in the ROM is accessible to all MPSs through a shared ROM bus. A centralized ROM is used because this method allows easy code sharing and maximum flexibility in the amount of code that can be dedicated to an MPS compared to a separate ROM for each MPS. The bus is arbitrated in a round robin scheduling discipline. When an MPS issues a ROM access, the ROM manager buffers the tag(s) associated with the sending MPS. The ROM manager also routes the microinstructions to the appropriate MPS(s) using the tag(s). Multiple tags are sent by an MPS when it is in an internal lockstep mode, and the same microinstruction is routed by the ROM manager to all MPSs involved in the lockstep. The ROM bus is also used to send interrupts.

There are two basic categories of interrupts: internal and external interrupts. Internal interrupts are sent between MPSs within a single RIC. External interrupts are sent between an MPS on one RIC to one or more MPSs on one or more other RICs. Each MPS has an interrupt manager. The interrupt manager sends and receives both internal and external interrupts. The interrupt manager gains control of the ROM bus to send an interrupt. The interrupt manager sends the following information: identification of the

interrupt source MPS, the destination MPS(s), the priority of the interrupt and run time information. An internal interrupt is sent to multiple MPSs to initiate a lockstep process or a pipeline process. The receivers of an interrupt respond as to whether the interrupt is to be immediately acted upon or not. An external interrupt is sent to the external interrupt manager. The external interrupt manager uses the priority of the interrupt to access a message block which is sent to external RICs and or other interruptable devices.

A process within the RIC is initiated by an interrupt. A process is defined as an instruction stream. An instruction stream is composed of microinstructions, macroinstructions or a combination of the two. Each process has a priority associated with it. Within a RIC there are 256 priority levels. The priority of a process and the priority of the interrupt which initiates this process have the same value. When an MPS receives an interrupt, its interrupt manager compares the received interrupt priority with that of the currently executing process. If the interrupt manager determines that the interrupt priority exceeds the current process priority, the interrupt manager signals that this interrupt process will cause a context switch. Otherwise, the interrupt manager indicates that the interrupt process priority is of lower priority. This type of feedback from the interrupt receiver to the sender is needed in the case of multiple receivers. If only a subset of the receivers can perform a context switch, MPSs would be idled unnecessarily while waiting for other MPSs to finish their higher priority process. If only a subset of the multiple receivers of an interrupt can perform a context switch, the interrupt is withdrawn and sent again later. This scheme prevents deadlock and unnecessary idling of resources.

If an interrupt is sent to a single receiver and the interrupt is of lower priority than the current process, the interrupt is buffered by the receiver MPSs scheduler. The scheduler buffers interrupts by priority in a 256 bit shift register. When a process is active, the scheduler scans through the shift register to find the process with the next highest priority. When the current process is finished or timed out, the scheduler uses the priority of the next highest priority process to access a table which contains a pointer to the process's context.

Programmable interconnect is used for routing the carry chain, the shift/rotate linkage, and the ALU result status flags. The routing of these signals depends upon the single or multi-chip structure being used. In the following, only the programmable carry chain is discussed. The carry-chain is unidirectional in nature flowing from MPS 0 to MPS 3, and then looping back to MPS 0. The carry routing logic is also responsible for handling carry in and carry out signals in the external and hybrid lockstep modes. The routing logic is expected to be implemented with pass transistors and is set up in a particular mode at the beginning of a structure by appropriate signals from the PLA, and remains set up that way until the next restructuring. For example, in the independent mode where all four MPSs are working as four independent processors, the routing logic isolates the carry chain into four

independent segments. In the internal lockstep mode the routing logic establishes a separate carry chain for each lockstep on the chip. Figure 8a shows the carry chain when MPS 1 and 2 are working in a lockstep, and MPS 0 and 3 are working as independent processors (one's complement arithmetic is used requiring the end around carry). Figure 8b shows the carry chain when all four MPSs are involved in four different external locksteps. Programming of the shift/rotate linkage and the ALU result status signals are similar. The ALU status signals differ slightly in that these signals from lockstepped MPSs are individually connected to a bus using a wired-AND configuration.

## 5. INTERNAL RAM

The internal RAM of the RIC is organized as four independent memory modules that are byte addressable. An NMOS RIC with a minimum geometry feature of one micron ( $\lambda$  equals .5 micron) could contain about 16-32K bytes of dynamic RAM. This RAM would occupy about one-third to one-half of the chip area. The internal RAM subsystem of the RIC includes four independent memory modules and a data bus interconnecting the RAM to the four MPSs. The data bus is designed to support the three basic internal structures of MPSs: independent, lockstep and pipeline. The memory subsystem also contains a memory mapper to automatically direct memory accesses to internal locations if the data is resident internally or to external locations otherwise. The memory subsystem is illustrated in Figure 9.

The data bus supports four concurrent accesses to memory, provided there is no interference between processors and memory. This bus allows a direct path from each MPS to its own memory module. When each MPS accesses its own memory module, then four simultaneous memory accesses can occur. If MPSs access memory modules other than their own, these memory accesses may result in memory interference, since a shared bus is used and multiple MPSs may access the same module resulting in queued memory requests. As shown in Figure 9, each memory module has a Memory Scheduling Unit (MSU) and a Bus Control Unit (BCU). When an MPS accesses its own memory module, it is directly connected through its BCU to its MSU. The MSU indicates whether there are pending memory requests or not. If there are no pending memory requests, the access occurs immediately. If there are accesses pending, the MSU queues a tag indicating the MPS which requested memory service. An MSU queues an MPS request with a first come first served scheduling discipline. When the MPS request reaches the head of the queue, the MSU signals this to the MPS. The MPS reissues its request and the memory access is performed immediately. When an MPS accesses a memory module other than its own, the BCUs are configured to make the connecting bus a shared bus, as shown in Figure 10a. The MPS first waits for access to the shared bus. The shared bus is scheduled by a round robin by demand discipline where the first MPS or memory module gets access to the bus in round robin order. After an MPS gains access to the bus, it sends the memory information and a destination tag indicating the destination memory

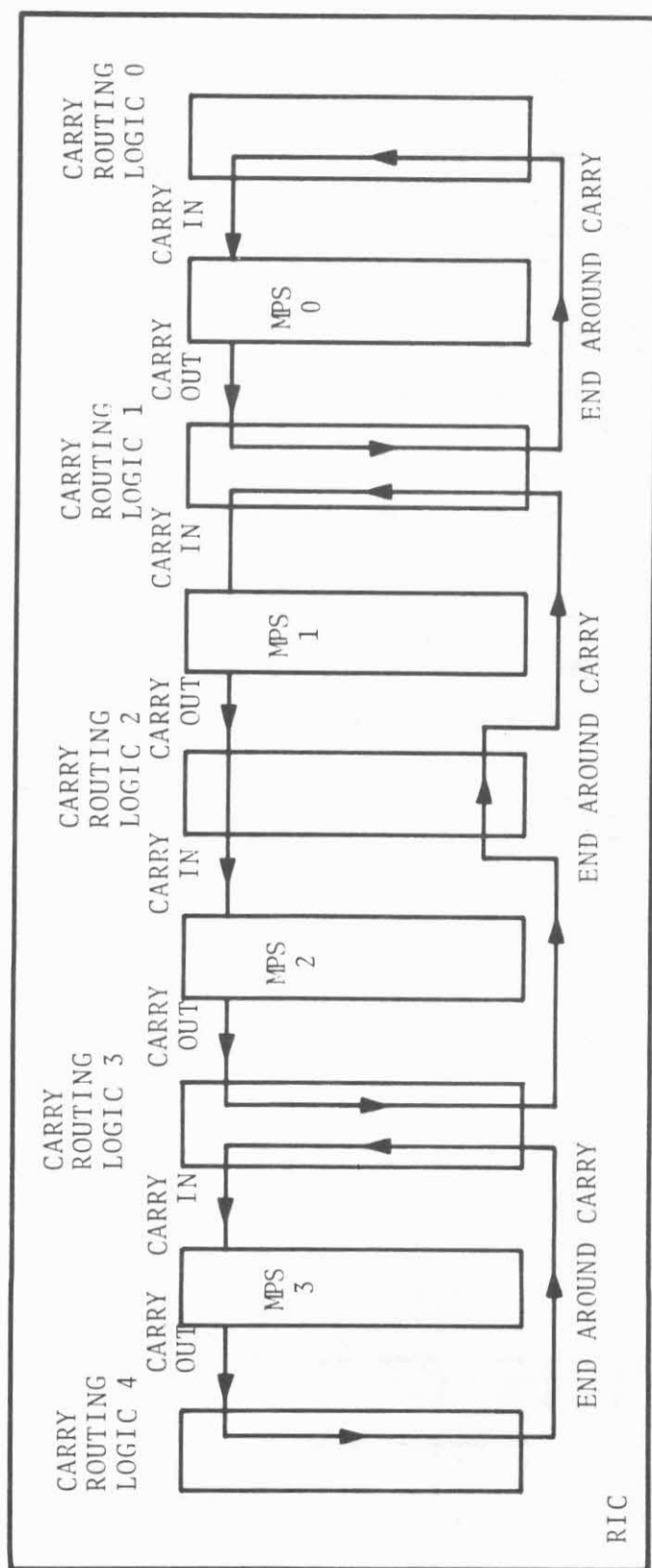


FIGURE 8A. Carry chain configuration when MPS 1 and MPS 2 are working in an internal lockstep and MPS 0 and MPS 3 are working as independent processors.



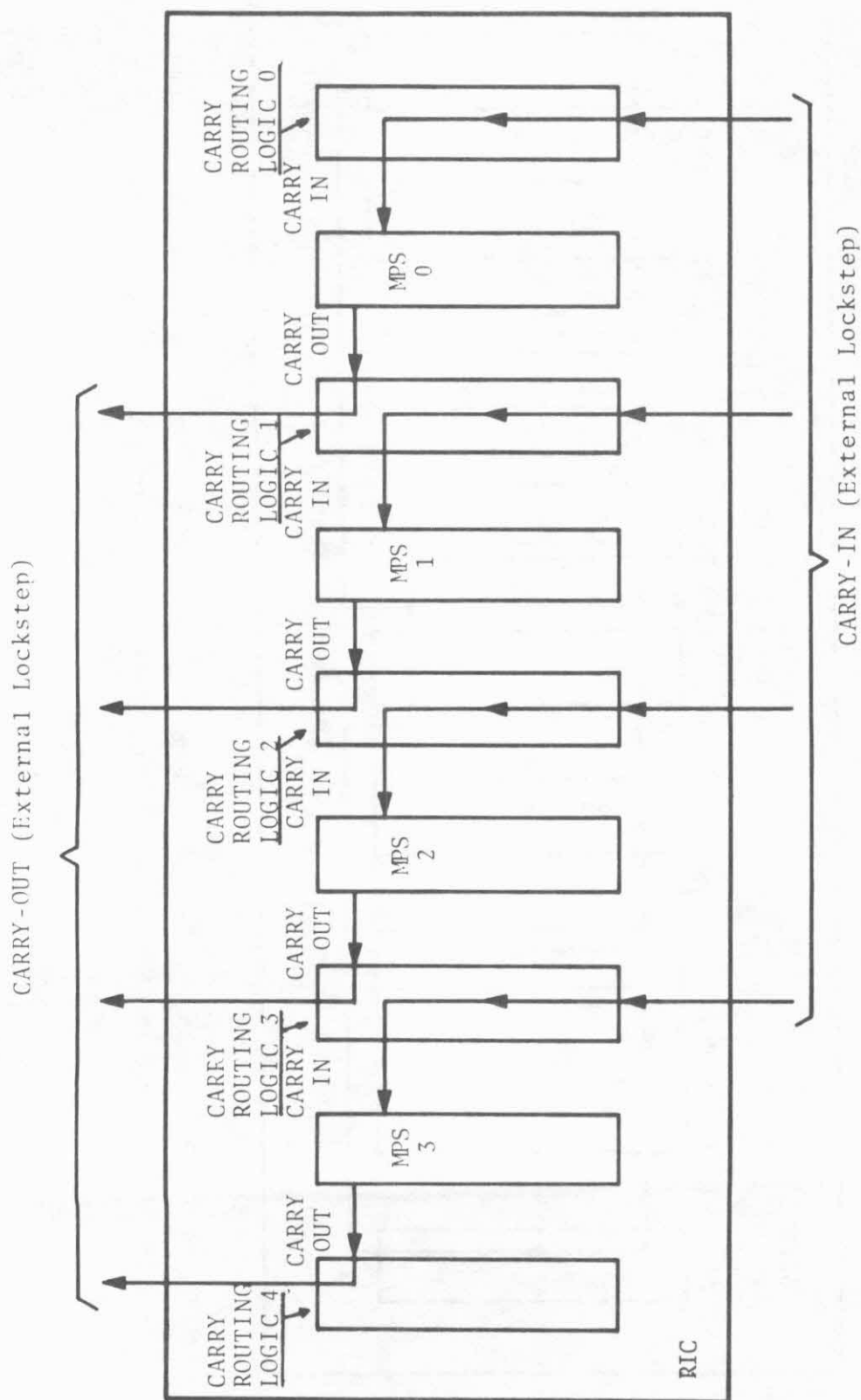


FIGURE 8B. Carry chain configuration when all 4 MPS's are working as independent processors.



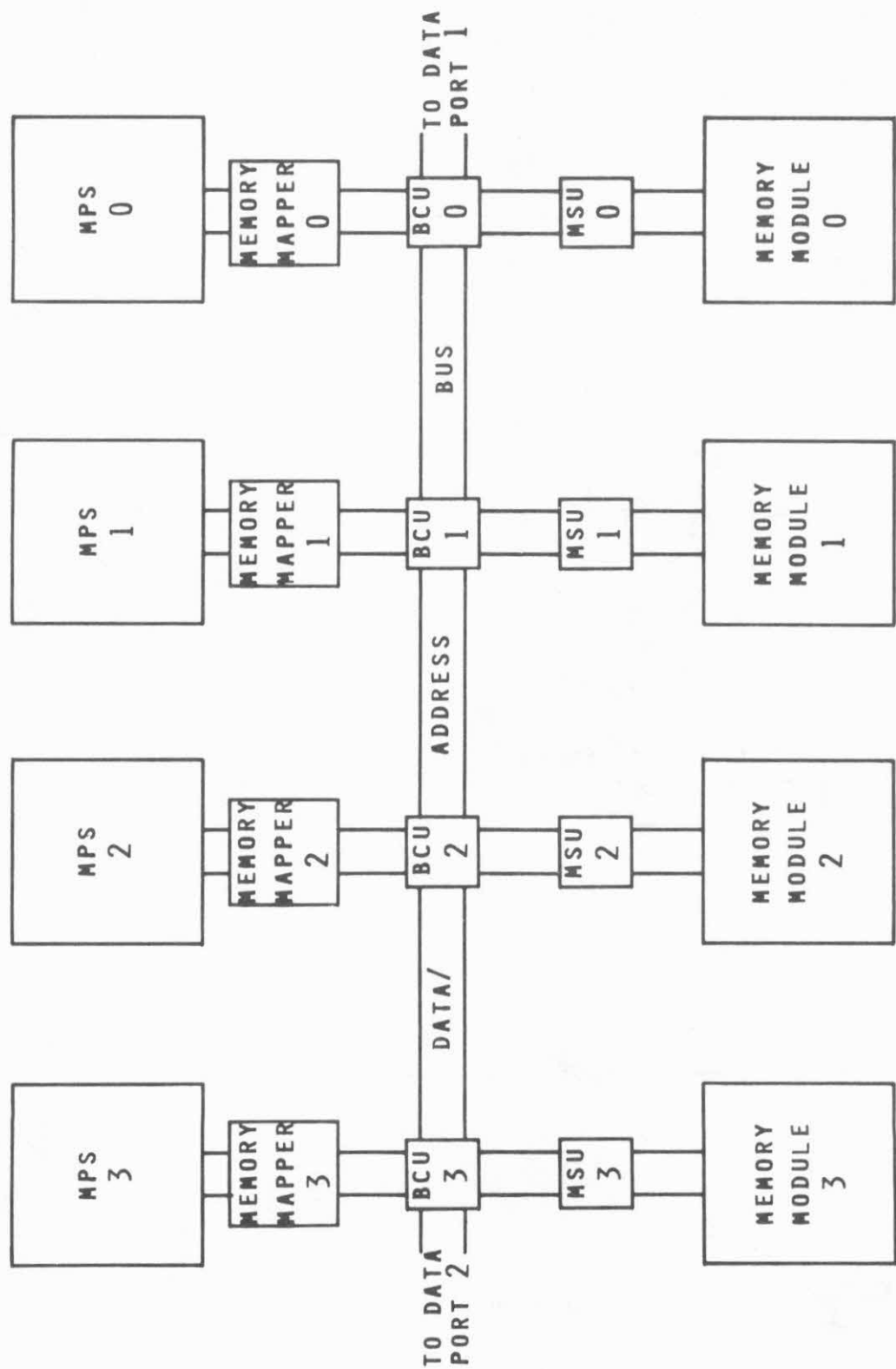


FIGURE 9 RIC RAM SYSTEM

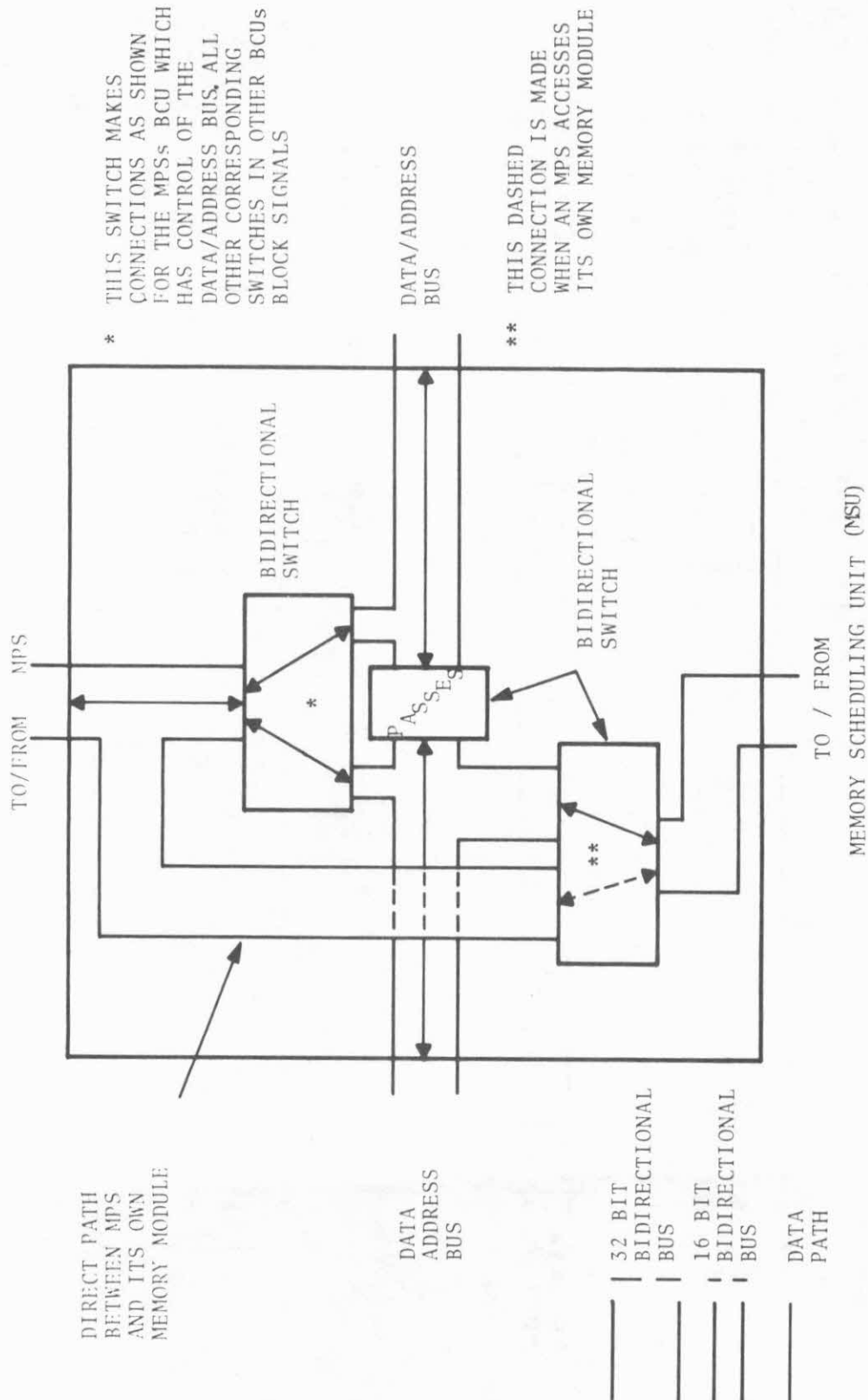


FIGURE 10A. BCU Configuration For Shared Data/Address Mode

module. The destination module sends the memory requests queue position on a separate bus. If 00 is sent the memory request is being processed immediately. Otherwise the two bit number indicates the number of pending requests before this current one. Any memory module can have at most four memory requests pending since an MPS can only have one memory request pending at a time. Each MPS has circuitry to monitor the bus. When the memory module that has an MPSs memory access pending completes a memory access, that MPS decrements the number of pending requests by one. When an MPS decrements this number to be zero, it means that its request is at the head of the queue at the memory module where this MPSs memory request is pending. When this MPS gains control of the shared bus, it reissues its request and the request is processed immediately. The operations described above support the memory accesses made by independent MPSs, lockstepped MPSs and pipelined MPSs.

The RIC memory system supports lockstepped and pipelined MPSs. Lockstepped MPSs can make simultaneous requests to their own memory modules. It is possible that lockstepped MPSs will not receive their request for memory service simultaneously because the queue length at one MPSs memory module could be different than the queue length at another lockstepped MPSs queue. Lockstepped MPSs are synchronized to avoid this problem. When lockstepped MPSs make a memory request, a wired-AND line which connects all MPSs in the lockstep is pulled low by each MPS. After each MPS has had its memory request serviced, it discontinues pulling this line down. When the last MPS has its memory request finished the line will rise to a logic one, indicating that the lockstep process can continue. Also lockstepped processors can access memory modules other than their own. In this case each lockstepped MPS would issue its request when it got access to the bus. The lockstepped memory access would be synchronized as above.

In addition to accessing memory, pipelined modes also use the data bus to send data between other MPSs in the pipe. Figure 10b shows the BCU configuration for pipelined data transfers. For this BCU configuration, the data bus is segmented to allow all adjacent MPSs in the pipe to transfer data in parallel, including transfers to MPSs on different RICs.

Each memory module is addressed with a 16 bit address. This allows for eventual growth of up to 64K bytes of directly addressable space for each of four MPSs. However an MPS supports two types of addresses: 16 and 32 bits. Sixteen bit addresses are used to directly access an MPSs own memory module. Thirty-two bit addresses are used to access other memory modules or external memory. In the case of accessing other memory modules, the most significant 14 bits are a tag indicating that the address is for an internal memory module. The next two significant bits select one of four memory modules. The remaining 16 bits point to an address in an internal memory module. If a 32 bit address does not point to an internal memory module directly it can either be an external address, or it can be a mapped address. A 32 bit address is mapped or external depending upon MPS control. If the address is designated to be an external address, it is sent to the external memory interface for processing. Otherwise, it is sent to the memory mapper. The memory mapper



uses an associative search to determine if the address is internal or external. If it is internal the associated internal address is sent to internal memory. If the address is external it is sent to the external memory interface.

## 6. EXTERNAL INTERFACE

The external interface for the RIC is designed to support multiple RIC configurations, interchip communication and data path communication between system memory and system I/O. Two versions of pin assignment are planned. An 82 pin version has two 16-bit data/address ports. A 114 pin version is the same as the 82 pin version except that it has two 32 bit data/address ports. The 82 pin RIC is discussed below.

In Figure 11 the RIC pin assignment is illustrated. There are five types of pin functions for the RIC: data/address, control, interrupt, status, and power/clock. The number of pins dedicated to each function group is listed in Table I.

Table I

FUNCTION	NUMBER OF PINS
data/address (2 ports)	50
control	2
interrupt	8
status (2 ports)	18
power/clock	4
	<hr/> 82

### Pin Assignment By Groups

#### 6.1. DATA PORT

The 82 pin version of the RIC has two 16-bit data/address ports. Each port has 16 bidirectional lines for carrying data and addresses. Associated with each port is a pair of handshake signals for gaining control of a shared

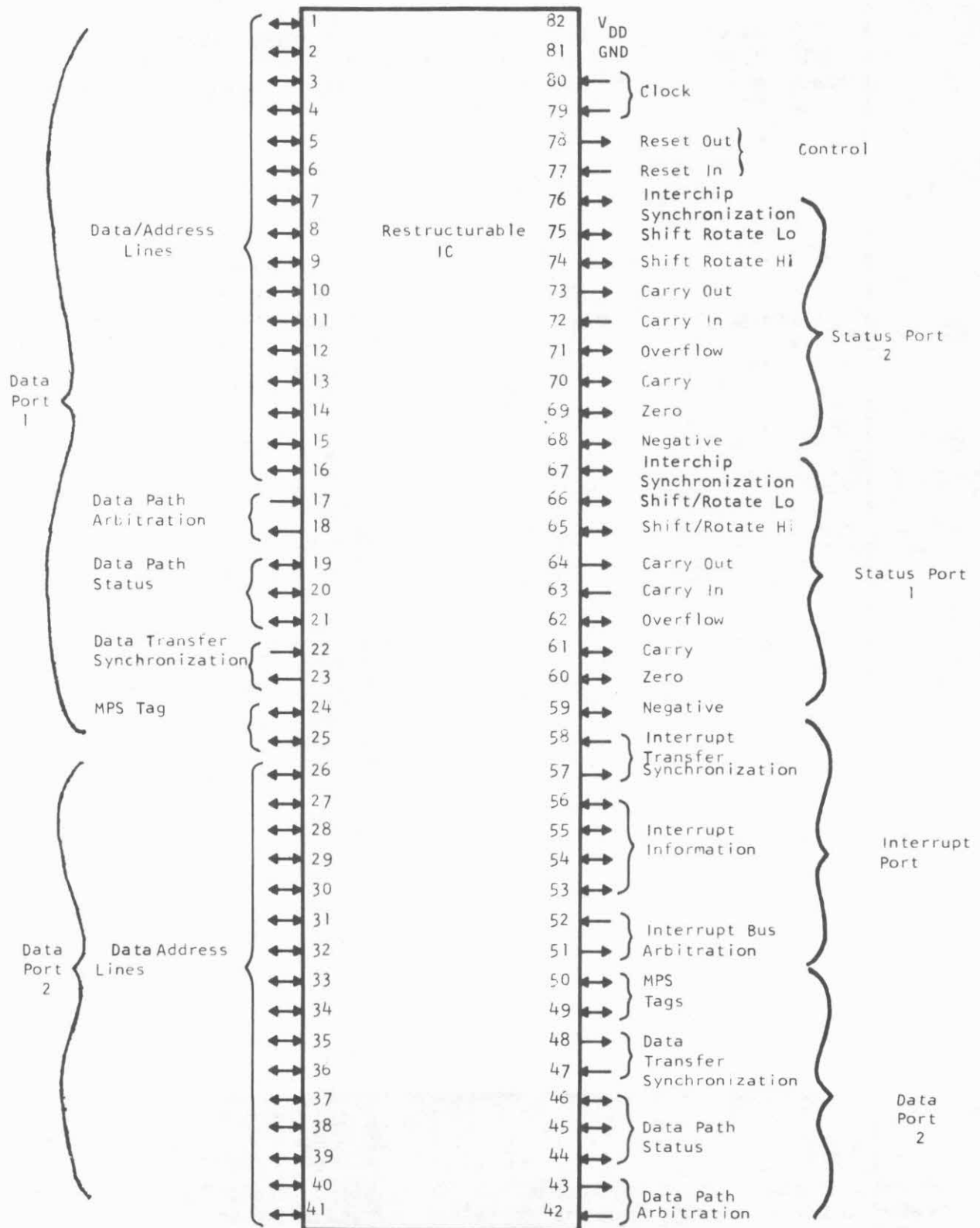


FIGURE 11. A Restructurable IC Pin Assignment

resources. The arbitration method for the shared resources is either round robin by demand, master-slave, or determined by external circuitry. The round robin and master-slave arbitration methods support a shared bus, while the external arbitration circuitry supports a network with a general topology. Each port also has a pair of signals for synchronizing the sending and receiving of data and addresses on the bus. Also, each port has a bidirectional set of three signals to indicate bus status. Four types of read/write operations are indicated by these three signals. The four operations are: access a user specified RIC, access system RAM, access system I/O, and access the resource whose destination address is sent at the beginning of the access. Finally each port has a bidirectional pair of MPS tag identifiers. The MPS tag identifiers are used to indicate the source MPS at the sender and/or the destination MPS at the receiver. The two data/address ports are independent. However, these two ports can be combined into one port by internally performing the same operation to both ports concurrently, and externally treating the two ports as one port.

## 6.2. STATUS PORT

There are two identical status ports. The status port's main function is to provide the signals to lockstep two MPSs on different RICs. Status port 1 can be used to lockstep MPS 0, MPS 1 or a lockstep of MPSs 1 and 0 to external MPSs. Status port 2 can be used to lockstep MPS 2, MPS 3, or internal locksteps including MPS 2 and/or MPS 3 to external MPSs. The use of the status ports is illustrated in Figure 6. A 64 bit wide external lockstep is formed with 4 MPSs on four RICs in figure 6. There are four pin functions in each status port: ALU result status, carry linkage, shift/rotate linkage, and MPS synchronization.

There are four ALU result status pins. These are: the Negative result status, N; the Zero result status, Z; the Carry result status, C; and the overflow result status, V. These four signals connect to a bus using a wired-AND configuration. This bus connects all externally lockstepped status ports. These signals are encoded to indicate up to one of 16 ALU result outcomes. The carry linkage is a carry-in signal and a carry-out signal. The carry-out signal of one RIC is connected to the carry-in signal of the next most significant RIC. The shift/rotate linkage is used to perform shift operations between externally lockstepped MPSs. The shift/rotate hi signal of a RIC is connected to the next most significant RICs shift/rotate lo signal. The shift/rotate hi signal of the most significant RIC is connected to the shift/rotate lo signal of the least significant RIC to provide the shift/rotate linkage. The MPS synchronization pin ensures that externally lockstepped MPSs are executing the same instruction in phase. Without synchronization, MPSs in an external lockstep can get out of phase because other MPSs on a RIC may be operating independently of the externally lockstepped MPSs. Thus the time to fetch a microinstruction may vary among the RICs containing lockstepped MPSs. The MPS synchronization pin serves as a flag

to indicate that each MPS has finished the previous instruction and has fetched the next microinstruction and is ready to execute it. The MPS synchronization pins are wired together in a wired AND configuration. When all externally lockstepped MPSs are ready to execute the next instruction, the MPS synchronization line will be high. If one or more MPSs is not ready, the line will be pulled low. When the MPS synchronization line is high, execution begins on the next clock cycle. (All RICs with MPSs in a common external lockstep must use the same system clock.) Shortly after execution begins, the MPS synchronization line is pulled low and it stays low until all MPSs are ready to execute the next instruction.

### 6.3. INTERRUPT PORT

The interrupt port serves two purposes. The first purpose is to receive and process interrupts in a manner similar to conventional microcomputers and microprocessors. The second is to provide for interchip communication. The interrupt concept has been generalized to include the capability to send interrupts to other receivers, providing for interchip communication. The purpose of interchip communication is to coordinate RICs to a task, to initiate a task, and to transfer information. The interchip communication system is used to transmit commands and/or small amounts of data. The bulk data part of an information transfer is communicated between memories. For example, a disc read operation is initiated by using the interrupt port of a RIC to send commands to a disc controller. The data transfer is accomplished on a separate data path between the disc system and the memory system. The interrupt port contains pins for arbitration, information and data transfer synchronization.

The interrupt port of the RIC has 8 pins. Two of the pins are used for arbitration of shared resources used during the sending of an interrupt. The same three arbitration modes used for the data ports are also used for the interrupt port: round robin, master-slave, or a general arbitration method.

Four pins of the interrupt port are dedicated to data transfer. The data protocol has minimal specification with maximal user definition. In the interchip communication mode, the first information sent on these pins is an address. The length of the address is designated by the user. When an interrupt is sent, all chips on a common interrupt bus receive the address and store it. The status signals indicate whether the information lines carry address or data. The receiver buffers the address portion as long as the status indicates address bits are being sent. After the destination address has been sent, each receiver uses the address to access a bit in the chips RAM to determine if this chip is an intended receiver of the interrupt. In the conventional interrupt scheme, the first information sent is the interrupt level. The remaining two pins of the communication port are used for interrupt bus status. The four status values are: sending address, sending



data, data/address nibble received, and interrupt information transfer completed.

An interrupt is sent by first gaining control of the interrupt bus. After gaining control of the bus, the interrupt data is sent. The amount of data that is sent is determined by the user. The hooks have been provided to send an optional destination address of variable length, a variable length data portion, and an optional source address of varying length as the components of the information sent during an interrupt. The interrupt information is buffered at the destination by the RICs external interrupt manager. The external interrupt manager interrupts the destination MPS and passes it the length of the message and a pointer to the interrupt message block.

#### 6.4. CONTROL LINES

There are two control lines, the Reset In (RI) and Reset Out (RO). The RI and RO signals from all RICs are connected together. RI signal is active high. When the RI signal is raised to a 1, the RICs begin to initialize themselves for operation. The RO signals are wired together in a wired AND configuration. When a RIC has completed the initialization operation, the RO signal which had been pulled low is allowed to float. When all RICs have completed initialization, the RO signal will be high indicating that the system has finished initialization.

#### 6.5. POWER/CLOCK

The RIC will use two power pins: +3 volts and ground. The RIC will use an on-chip clock generator. This will allow a crystal to be placed across the two clock inputs, or an external clock can replace the crystal.

#### 7. CONCLUSION

The RIC has been described at a high level and many details have not been included. The major goal of the RIC is the achievement of a highly flexible part that can be used to achieve a wide variety of specific hardware designs through programming. The designed-in flexibility of the RIC provides for this programming. The flexibility within the RIC includes: user definable micro language and assembly language, user programmable microcode, dynamic coordination of multiple internal processors, coordination of processors on

multiple RICs, internal memory that can be used either as a caches or as an element of a virtual memory hierarchy, general topology for interchip communication and external data paths and a user definable interrupt mechanism.

#### REFERENCES

- 1 R. G. Arnold, and E. W. Page, A hierarchical, restructurable multi-microprocessor architecture, 3rd Annual Symposium of Computer Architecture, pp 40-45. 1976
- 2 S. I. Kartashev and S. P. Kartashev, A multicomputer system with dynamic architecture, IEEE Trans. Comput., vol. C-28, pp. 704-720, October, 1979.
- 3 Bipolar Microcomputer Components Data Book, Texas Instruments Inc.
- 4 The Am2900 Family Data Book, Advanced Micro Devices Inc.

## Communication in a Tree Machine

*Sally A. Browning*

Bell Laboratories  
Murray Hill, New Jersey 07974

*Charles L. Seitz*

Computer Science Department  
California Institute of Technology  
Pasadena, California 91125

### *ABSTRACT*

Communication assumes a progressively dominant and limiting role in VLSI because it becomes relatively more expensive in chip area, signal energy, and time. The principle of locality becomes all important to integrated systems design, and implies that larger single processors are not the route to performance improvements. One computer architecture that can exploit the capabilities of VLSI is an ensemble of small processors operating concurrently.

The tree machine is such a structure. Each of the many processors in the binary tree can communicate directly only with its parent and two children. However, the tree is programmed as if each processor had an arbitrary number of descendants, and the programs are compiled into code for a binary tree. We describe the communication structure of tree machine programs, the compilation process, and the underlying hardware.

Jan 30, 1981

## Communication in a Tree Machine

*Sally A. Browning*

Bell Laboratories  
Murray Hill, New Jersey 07974

*Charles L. Seitz*

Computer Science Department  
California Institute of Technology  
Pasadena, California 91125

### 1. The Tree Machine Architecture and Project

As Very Large Scale Integration (VLSI) becomes a reality, we have the opportunity and motivation to build entirely new kinds of computers. The traditional view of a single large processing unit physically separated from a single large store by a memory bus is certainly realizable in VLSI. But, as Sutherland and Mead<sup>5</sup> have observed, this architecture requires too much global communication to be a good fit to VLSI. It is also a carryover from the era in which processing and storage were best implemented in different technologies.

Most of the space on an integrated circuit is occupied by the wires that carry control and data to the functional blocks. It is no longer sufficient to spend most of the design effort on the individual cells and leave the wiring until the end. Rather, the interconnection must be considered an integral part of the design. As Seitz<sup>4</sup> has pointed out, making larger scale single processors in VLSI scaled to submicron dimensions becomes self-defeating: the diffusion delay in a wire scales up quadratically while the delay of a MOS switch scales down linearly. Thus, the ratio of communication to switching delay scales up as the third power of the scaling factor, and rapidly becomes a dominant design factor for metal sizes below about one micron. Faster signal paths on higher, thicker layers of wider metal runs are possible and a likely feature of new technologies, but will be a limited resource and will require larger drivers built on the diffused, polysilicon, and thin metal layers.

Together, these observations might be called the principle of locality for VLSI. If a signal must travel between two widely spaced points, it will either go slowly, or require extra area for bigger drivers and thicker wires. The incentive to design chips with local communication is strong.

The tree machine architecture<sup>1</sup> provides a general purpose computing environment while capitalizing on the properties of VLSI. The tree machine is a collection of very small processors connected together as a binary tree. The processors are identical, and each has its own memory. There is no global communication, only communication between parent and child in the tree, and between the root of the tree and the external world. This architecture gives rise to integrated circuits that have regular interconnect, local communication, and many repetitions of a single processor design. These integrated circuits, in turn, can be assembled in regular patterns at the printed-circuit board and backplane level to construct machines with thousands to hundreds of thousands of processors.

The tree machine is general purpose because it is programmable. A varied collection of algorithms have been designed that take advantage of the available concurrency. In each case, the time complexity of the algorithm is roughly equal to the number of data elements in the problem. Thus, sorting can be done in linear time, and matrix operations in  $O(n^2)$  time. Many graph problems, including some that are NP-complete, can be solved in  $O(\text{edges})$  time, provided the tree machine has enough processors. Problems that require an exponential growth in resources, like the NP-complete problems and divide-and-conquer algorithms, are a good fit for the tree machine architecture: each additional level in the tree doubles the number of available processors.

This paper describes work done during the summer, 1980, at Caltech. The first design has one processor per chip; As processing with finer design geometries becomes available several processors and their memories will be put on a single chip. The goal is to have a tree machine built and tested by the fall of 1981, and a machine of at least 1023 processors working in 1982.

A feature of the tree machine that makes it particularly attractive for implementation in a research environment is that it can be viewed as a recursive structure. Each node is the root of a subtree. Thus, once the root of the tree has been tested, it can test, recursively, the rest of the tree. Once a bootstrap program has been loaded into the root, it can load the other processors. The importance of simple testing and loading procedures increases with the number of transistors on an integrated circuit.

## 2. Programming a Tree Machine

Tree machines are programmed in a high level language resembling Hoare's "Communicating Sequential Processes" (CSP) notation.<sup>3</sup> The notation, TMPL, is described in full elsewhere;<sup>1</sup> here we will look only at the communication primitives. This paper describes the tree machine communication protocol, and the mapping between the programmer's view of communication and the hardware. Thus, it is in large part a discussion of the TMPL compiler, whose major parts are shown in Figure 1. TMPL programs are written for trees with arbitrary fanout, and transformed by the compiler into source code for a binary tree; this is the MAP step, and is described in section 3.



Figure 1. TMPL Compilation Process

While TMPL supports four communication primitives, the hardware implements only two of them. We remove the other two in the QUEUE step. This mapping is discussed following a presentation of the hardware implementation. The final two compilation steps are familiar: the source code is translated into machine code in the COMPILE step, and a load stream is composed in the ASSEMBLE step. These two processes are not described. We begin with a brief presentation of the notation, concentrating on the constructs used in communication.

### 2.1. Communication Primitives in TMPL

The basic building block of TMPL is the processor definition. Each definition describes a self-contained computational unit that communicates through ports with other processors. A processor definition includes the naming of one *external port* to its parent in the tree, and an arbitrary number of *internal ports* to descendents. Communication statements mention port names through which the message will pass, rather than naming target processors. As a result, processor definitions are written without regard to the eventual connection plan of the tree. A processor expects to follow a specific communication protocol when accessing a port. Any processor that follows the same protocol can be connected to the other end of the port.

This definitional locality makes possible a parts kit of standard processor definitions. Each part is a processor, or tree of processors, that can be completely characterized by the behavior at its ports. As long as the expected messages are sent and received, the parts will work anywhere in the system.

Inter-process communication can be specified in two ways, either as an imperative statement, or as a conditional expression. The statement form results in the processor being blocked until the communication is successfully completed. The conditional form appears as part of a loop or case statement, and is performed only if both processors communicating along the specified port are ready to exchange the message.

Syntactically, message statements and expressions are identical; the general form is shown below. We retain Hoare's notation for the direction of the communication: ? indicates input, and !

is used for output.

$$\left[ \begin{array}{c} \text{port} \\ \text{list of ports} \end{array} \right] \left[ \begin{array}{c} ? \\ ! \end{array} \right] \text{message}(\text{arguments})$$

They are distinguished by context: message statements can occur wherever a statement is valid, while message expressions are legal only in guards, described below.

A communication involves two processors connected via a port. An output request to the port from one processor must match up with an input request for the same message from the other processor in order for the communication to take place. Either the output or the input can be done conditionally, but not both. This restriction prevents kind of deadlocking: the "after you, after you" situation that arises when neither processor will commit itself to the conditional exchange.

TMPL provides compile-time checking for occurrences of the illegal conditional-to-conditional communication by requiring that message names be typed. These types specify how the message will be used: **imp** for imperative mode, and **cond** for conditional. The message names, types, and directions (input or output) are made externally available. When the tree connection plan is read and the processors linked through ports, the message interfaces are compared. Illegal communications, as well as messages that cannot be paired up, are flagged.

For example, suppose we have three processor definitions called A, B, and C. Message ports, names, types, and directions used in the three processors are shown below, in Figure 2.

Processor	Port	Port Type	Name	Type	Direction	Arguments
A	parent	external	load	cond	?	1
	parent	external	unload	cond	!	1
	left	internal	load	imp	!	1
	left	internal	unload	imp	?	1
	right	internal	load	imp	!	1
	right	internal	unload	imp	?	1
B	parent	external	load	cond	?	1
	parent	external	unload	imp	!	1
C	parent	external	load	cond	?	1
	parent	external	unload	cond	!	1
	left	internal	load	cond	!	1
	left	internal	unload	imp	!	1

**Figure 2. Message Descriptors**

If we connect the left port of A to B's port, we find that A will send *load* messages, and B will receive them; similarly, B will send *unload* messages and A will receive them. In addition, all communications between the two are either imperative to imperative or imperative to conditional, and thus valid. A connection between the left port of C and B produces invalid communications. While C outputs *load* messages and B receives them, both are doing it conditionally. And the *unload* messages have the same direction: both processors want to send the message. Thus, B and C cannot be connected using the left port of C. Incidentally, the protocols match up if B and C are connected through their parent ports, but in order to preserve the tree structure, connections must always be made from an internal port to an external one, that is, from parent to child.

## 2.2. Message Statements and Expressions

Message statements are a familiar concept, resembling subroutine calls: the processor executing the statement is blocked from further execution until the communication is successfully completed. Message expressions, however, are more complicated. The following paragraphs describe them in more detail, beginning with a description of the statements in which the message expressions may

appear.

As in Dijkstra's notation,<sup>2</sup> TMPL has generalized loop and conditional statements made up of a set of guarded commands. A guard is an expression; the command will be executed only if the guard has the value **true**. A TMPL guard can be a logical expression, a message expression, or a combination of the two. A single guard can contain at most one message expression. Each of the following lines contains a valid guarded command.

$$i < 9 \text{ AND } i \geq 0 \rightarrow j := j * 10 + i$$

$$\text{NOT found AND } p?arc(i,j) \rightarrow \text{found} := \text{start} = i \text{ AND } \text{end} = j$$

$$\text{left, right?answer} \rightarrow p!answer ; \text{count} := \text{count} + 1$$

The syntax for loop and conditional statements differs only in the braces used to enclose the set of guards: curly ones are used for loops, and square ones for conditionals. Semantically, however, the two statements are quite different. A loop statement is executed repetitively until all guards are false. A conditional statement is executed exactly once: at least one guard must be **true**, otherwise the statement and the program will abort. In both statements, if more than one guard is **true**, a nondeterministic choice will be made. For example, if  $i=5$  when the conditional statement below is entered, all three of the guards are true. We cannot assume that the first one will be chosen, but must settle for a random choice.

$$\begin{aligned} &[ i \leq 9 \rightarrow \text{SKIP} \\ &| i > 0 \rightarrow i := -9 \\ &| i = 5 \rightarrow p!five ; \text{found} := \text{found} + 1 \\ &] \end{aligned}$$

The nondeterministic property of the loop and conditional statements is an integral part of programming a tree machine. The processors act independently; a pair of them is synchronized by communication. Because a processor has three ports to other processors and knows nothing of the timing characteristics of its neighbors, a TMPL program can describe only the *sequence* it will use to access the ports. In many cases, a message could arrive on any one of a set of ports, giving rise to a set of guarded commands, each one triggered by communication on a specific port. For example, the conditional statement below has message expressions for all three ports. We can make no statements about the order in which the guards become true: it depends both on when the ports become active, and on the choice between **true** alternatives.

$$\begin{aligned} &[ \text{bus, left, right?done} \rightarrow \text{active} := \text{active} - 1 \\ &| \text{left, right?notDone} \rightarrow \text{active} := \text{active} + 1 \\ &] \end{aligned}$$

Conditional and loop statement guards are the only place that message expressions may appear. They may be combined with the familiar logical expressions to make more complicated expressions. While an individual guard can contain at most one message expression, a guard set can contain many, and can mix message expression guards with those containing only logical expressions. If logical and message expressions are combined in a guard, the logical expression must appear first, and is evaluated first. Because the message expression can have the side effect of doing an input or output as it is evaluated, it is examined *only* if the value of the logical expression is **true**.

Like logical expressions, message expressions are evaluated. Unlike logical expressions, they can assume one of *three* values. **False** signifies that the communication cannot be done, and is assigned whenever some *other* communication is pending on the selected port. A message



expression is **true** if the communication is possible: the port is waiting for exactly this message. The **maybe** value is assigned to the expression if it is neither true nor false, that is, the port mentioned in the message expression has no pending communications.

The key to evaluating a message expression is the restriction that at least one of the two processors involved in a communication must use the imperative form. The message *statement* forces the communication, making the issuing processor and associated port busy until the transfer has been completed successfully. Thus, a processor that is evaluating a set of message *expressions* has only solid, imperative communications to match up with. The state on the busy ports won't change while a decision is being made.

### 3. Mapping Arbitrary Fanouts Onto a Binary Tree

TMPL programs are written as if the processor had as many immediate descendents as needed. In fact, the underlying architecture is a *binary tree*, and arbitrary fanouts are simulated. A processor definition that declares more than two internal ports to children becomes the root of a *composite* processor, shown in Figure 3. Several layers of the tree are used to provide the required number of descendents, and the intermediate processors, called *padding* processors, are provided with a skeletal program that allows them to pass messages between the parent and children. The process of generating those skeletal programs is the subject of this section. We begin with some guidelines for the mapping algorithm, look at a pair of *routing numbers* that can uniquely identify a descendent processor, and finish with a case by case treatment of the seven kinds of communication that the padding processors must handle.

#### 3.1. The Mapping Constraints

The constraints we place on the mapping algorithm have distinctly different flavors. The first one reflects the underlying system architecture, while the second constraint is an arbitrary design decision.

First, there is no wildcard message name that matches everything. Because message expressions must eventually be evaluated as **true** or **false**, the hardware knows about message names. Thus, each padding processor must be tailor-made for the problem and will handle only those messages that can pass between parent and child. If the parent and children exchange *load*, *arc*, and *answer* messages, the padding processor must have message statements or expressions for *load*, *arc* and *answer* messages as well.

Our second constraint is an attempt to minimize the amount of information the mapping algorithm needs: we allow the program for the parent processor to be modified in the mapping process, but not that of the child. It is clear that the code in the composite processor will require modification: it specifies communications on more ports than it really has. But it is not necessary to know at compile time what is connected to the ports.

Several benefits arise from this constraint. We can compile a set of padding processors based only on information available in a single processor definition. The resulting composite processor has precisely the same communication characteristics as the unmapped processor. Thus, assertions about the communication properties of the original processor hold for the composite.

#### 3.2. The Routing Numbers

We use two integers, a *depth finder* and a *path*, to uniquely address a specific descendent of a composite processor. The depth finder indicates whether or not the message has reached a "leaf" of the composite. The path selects the left or right branch at each processor in the composite. New values for the depth finder and path are calculated at each level in the composite processor.

The depth finder measures the number of padding processors that must be traversed to reach the bottom of the composite processor. Thus, the depth finder is not a measure of distance from the root; rather, it is the distance from the leaves. Suppose we are simulating a fanout of  $n$ . The composite processor will contain  $n-1$  processors: the root of the composite supplies two connections

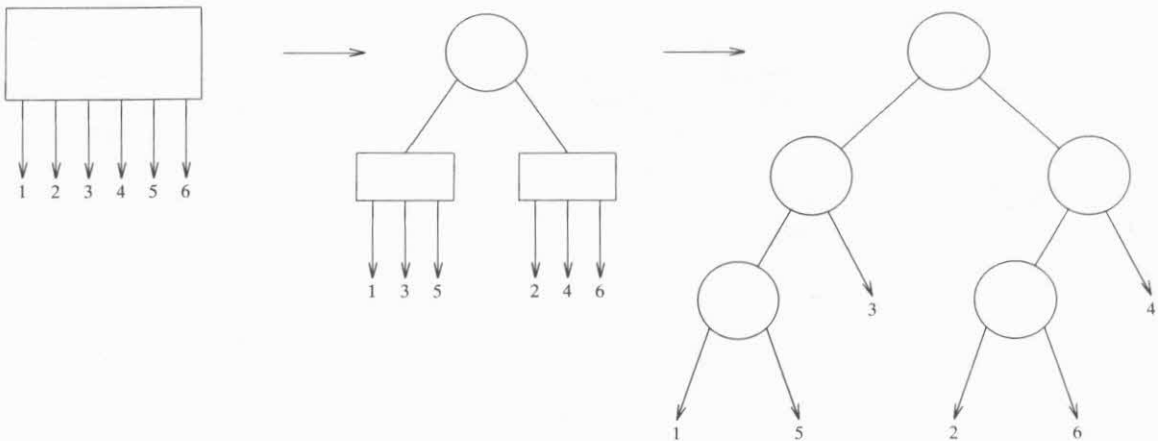


to children, and each additional padding processor adds two connections while occupying an existing one, for a net gain of one connection. Thus, a composite processor with fanout  $n$  contains one copy of the modified parent processor and  $n-2$  padding processors. As we descend in the composite processor this count can be recalculated to describe the number of padding processors in the composite if the current one is the root. At the leaves of the composite, the depth is zero.

The rule for calculating the depth finder value depends on the fact that unbalanced composites are heavy on the left side, as shown in Figure 3. The algorithm for calculating the depth finder is given below. It is applied recursively for all subtrees in the composite processor: each descendent is the root of a subtree, and supplies the value for  $depth_{parent}$  to the next level.

$$\begin{aligned} depth_{parent} &= n-2 \\ depth_{left} &= \frac{depth_{parent}-1}{2} \\ depth_{right} &= \frac{depth_{parent}-2}{2} \end{aligned}$$

The second argument specifies a path to the desired child, and relies on the allocation of logical to physical ports shown in Figure 3. The numbering of logical ports starts with 1, with odd ports assigned to the left side, and even ones to the right. This rule is applied recursively until all leaf processors have binary logical and physical fanout.



**Figure 3. Mapping Arbitrary Fanouts onto a Binary Tree**

Suppose the processor has a logical fanout of  $n$ , and wants to communicate with its  $i^{th}$  child. The path is initially set to  $i$ . At each node in the composite, odd path values are passed to the left, and even values to the right. A new path value is computed at each level according to the algorithm below. This, in conjunction with the depth finder, can be used to uniquely select a descendent.

$$\begin{aligned} path_{parent} &:= i \\ [ \text{ odd}(path_{parent}) \rightarrow path_{left} &:= \frac{path_{parent}+1}{2} \\ | \text{ even}(path) \rightarrow path_{right} &:= \frac{path_{parent}}{2} \\ ] \end{aligned}$$

Two routing numbers are needed to locate a specific processor because of the asymmetry of the composite processor. Paths from parent to child are not always the same length. In Figure 3, for example, it takes three steps to reach the sixth child, and only two to find the third one. Both the identity and depth of a descendent are essential information, and cannot be encoded in a single number.

### 3.3. Generating Programs for the Padding Processors

Padding processor programs are generated based on the parent's view of communication. We do not have access to the programs for the descendent processors, and must retain the original message interface in the fully padded composite. A list of message statements and expressions that the parent program might utilize follows:

1. imperative output
2. imperative broadcast output
3. imperative input
4. imperative broadcast input
5. conditional output
6. conditional input
7. conditional broadcast input

Note the absence of conditional broadcast output, better described as the case where all descendents from a node are ready to input the same message. This state can be expanded into a guard that is the logical AND of a set of messages expressions, one for each descendent. Since guards cannot contain more than one message expression, we do not implement conditional broadcast output.

Each of the valid cases is discussed briefly below. The general strategy is this: the parent processor definition is modified to communicate with two descendents through internal ports *l* and *r*. Two, one, or none of the routing numbers are appended to the message, as needed. There may be extra messages used for synchronization between parent and child: the message no longer travels directly.

Each padding processor is given the program segment required to read or write the messages expected by the parent. If any routing numbers have been added to a message, they must be stripped off before being handed to the child processor: it is expecting the original message. The depth finder, which records the number of padding processors yet to traverse, is zero at the bottom of the composite processor.

Imperative output directed at a specific processor requires both routing numbers to locate the receiver. Imperative broadcast output communication requires no additional arguments or messages. The message is allowed to spread throughout the composite processor, since it is directed at all descendents.

Imperative input from a specific processor requires both routing numbers, like the corresponding output command. In fact, any time a specific processor is mentioned in a communication statement, both numbers are used to locate it. Imperative broadcast input asks for a message from any one of its children. Note that the statement `c(*)?msg` is equivalent to a conditional statement:

```
[ c(1)?msg → SKIP
  | c(2)?msg → SKIP
  . . .
  | c(n)?msg → SKIP
]
```

Remember that guards are not evaluated in any particular order: a non-deterministic choice is made among those that evaluate to **true**.

The imperative broadcast input is treated as if it were a conditional statement: each child is

asked whether it has something to give the parent. If not, the next child is interrogated until one is found that is trying to send the matching message. The fact that the input is imperative guarantees that there will be at least one child that wants to send the message.

The template for imperative broadcast input relies heavily on the fact that message expressions are evaluated. The pair of statements that follow demonstrate a technique for finding out how a message expression was evaluated.

```
got := FALSE ; { NOT got AND !?msg → got := TRUE }
```

The loop will execute zero or one times, and the boolean value *got* can be used to find out what happened. The loop will be stuck evaluating the guard until some communication is initiated on the port called *l*. If the communication is a request to output *msg*, the value of the guard is TRUE, and *got* becomes true. The loop will not be executed again because of the *NOT got* phrase. If, on the other hand, the communication initiated on port *l* does not match, the value of the guard is false, and the body of the loop is never executed; *got* remains false. Thus, the value of *got* tell us whether or not a *msg* was input from port *l*. Imperative broadcast input asks each child in turn to output a message. As soon as a child is found that will satisfy the request, the polling terminates.

Conditional input, conditional output, and conditional broadcast input are all similar. In each case, the specific child (or each child in succession) is asked whether or not it can supply the requested message. If so, the message is accepted and moved up to the top row of padding processors so that it is available for the parent. The padding tree will have only one instance of a particular message stored in it at a time. If the guard is part of a loop, the message will be re-requested before the next iteration of the loop.

Conditional input and output require the use of both routing numbers, but do not send up an explicit *no* answer. Conditional broadcast input uses the *no* answer to continue polling the children, but needs only the depth finder routing number.

Appendix 1 contains templates for generating skeleton programs for the padding processors. The compiler uses these templates, filling in specific port and message references as needed. The padding processor program is a conditional statement, with a guard for every possible communication between parent and child.

#### 4. Communication Primitives in the Hardware

We devised two criteria for the tree machine processor and port design. First, the design must use chip area sparingly. In general, we are willing to sacrifice execution speed in order to limit the number of functions built into the hardware. Each new instruction is carefully scrutinized before being added to the repertoire. Second, the processor should reflect the structure and scope of TMPL. We intend to program this machine solely in TMPL. Thus the instruction set is designed for easy translation from TMPL to machine code.

These two goals compliment each other when designing instructions to implement TMPL statements that do not involve communication. However, when we attempt to implement directly the rich communication structure of TMPL, we find the complexity of the processor, and thus the area it occupies, increasing.

The double queue implementation discussed below is a nice compromise. It provides direct implementation of two of the four communication modes, conditional input and imperative output. The other two modes can be implemented via compiler translation into the chosen modes. This transformation is discussed later.

##### 4.1. An Overview of the Queue Design

A TMPL port is a bidirectional one: the same port is used for both input and output. The underlying hardware has two distinct unidirectional connections between the processor, controlled by a port. The hardware port is a simple micro-coded processor, distinct from the tree machine processor. The port and its instruction set are described in more detail in the next section.

Each unidirectional connection between processors has an associated queue that buffers messages, thus decoupling the two processors, as shown in Figure 4. The length of the queue determines the independence of the communicating processors, but cannot matter to the algorithms controlling the port. One may assume a length of one for all of the discussion that follows. A queue size will eventually be chosen through experimentation and simulation.

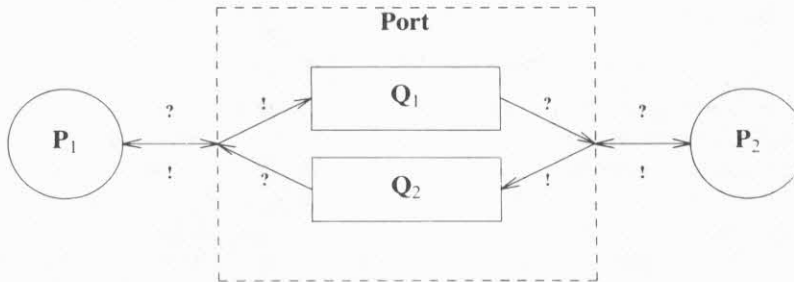


Figure 4. Communication Using Queues.

Each access to a port causes one word in the associated queue to be read or written. Thus, a TMPL message with several arguments is compiled into a sequence of messages, one for each argument:

$!!\text{arc}(i,j)$  becomes  $!!\text{arc}; !!i; !!j$

$!?\text{arc}(i,j) \rightarrow$  becomes  $!?\text{arc} \rightarrow !?i; !?j$

Notice that the message name is no longer associated with its arguments. If the two communicating processors have different notions of how many arguments the message contains, chaos will ensue. An alternative implementation would tag each argument with the message name, as in the example below.

$!!\text{arc}(i,j)$  becomes  $!!\text{arc}(i); !!\text{arc}(j)$

Tagging provides some measure of run-time validity checking on messages, at the cost of extra bits in each entry in the queue. Since we have complete information about the nature of the communication between processors at *compile* time (see Figure 2), we do better to put the checking there, retaining our streamlined port design.

#### 4.2. The Port Instruction Set

The port responds to three instructions from the processors it interfaces. **MATCH(msg)** compares its argument with the first element in the queue. It returns **true** if they match, **false** if they are different, and **maybe** if the queue is empty. **READ** removes the first element from the queue and returns its value to the requesting processor. If the queue is empty, it waits until something is inserted. **WRITE(msg)** adds its argument to the end of the queue, returning **true** to the writer. If the queue is full, **maybe** is returned, and the writer must retry the operation. Figure 5 formally presents the algorithms. We assume the usual operations on queues (called  $Q$  in Figure 5):  $\text{empty}(Q)$  is a boolean function that is **true** when the queue is empty,  $\text{head}(Q)$  returns the value of the first element in the queue,  $\text{removeHead}(Q)$  returns the first element and removes it from the queue,  $\text{full}(Q)$  is a boolean function that is **true** when the queue is full, and  $\text{add}(Q, \text{msg})$  adds  $\text{msg}$  to the end of the queue.

It remains to show how TMPL message statements and expressions are compiled into port instructions. We will look only at conditional input and imperative output. The next section describes a technique for transforming the missing communication modes into these.

Conditional inputs require the **MATCH** and **READ** instructions. **MATCH** is used to satisfy the condition: the message will be input only if **MATCH** returns **true**. The arguments are retrieved with **READ** instructions.

```

{ MATCH(msg) →
    [ NOT empty(Q) AND head(Q)=msg → return(TRUE)
    | NOT empty(Q) AND head(Q)≠msg → return(FALSE)
    | empty(Q) → return(MAYBE)
    ]
| READ →
    [ NOT empty(Q) → return(removeHead(Q))
    | empty(Q) → SKIP
    ]
| WRITE(msg) →
    [ NOT full(Q) → add(Q,msg); return(TRUE)
    | full(Q) → return(MAYBE)
    ]
}

```

Figure 5. The Port Instruction Set

$$p?arc(i,j) \rightarrow \text{becomes} \quad MATCH(arc) \rightarrow READ; i:=READ; j:=READ$$

Imperative outputs use only the WRITE instruction, but are complicated by the fact that the output will fail if the queue is full. Thus, they are implemented in a loop, shown in the TMPL notation below.

$$p!arc(i,j) \rightarrow \text{becomes} \quad \begin{aligned} & [ WRITE(arc) \rightarrow SKIP ]; \\ & [ WRITE(i) \rightarrow SKIP ]; \\ & [ WRITE(j) \rightarrow SKIP ] \end{aligned}$$

The first implementation of the port utilizes queues of length two. The port is split between the two processors that share it, with one word of each queue in each processor. Transfers between the two halves of the queue is done bit-serially, with one bit transferred during each storage cycle.

#### 4.3. Other Comments about the Processor

The processors in our present design have 12-bit registers, and a 12-bit address for accessing a program store organized as 4-bit nibbles. Instructions are variable length, ranging from two to six nibbles long. Most data is stored in the sixteen general purpose registers. Floating point numbers occupy four registers: one for the exponent, and three for the mantissa. A subroutine for floating point multiply is about 150 nibbles long and executes in about 600 storage cycles.

This relatively serial organization and direct connection to on-chip storage allows a short storage cycle and simple instruction pre-fetch techniques. Internal pipelining in the pre-fetch organization means that some nibbles fetched are not executed but this cycle is used to good advantage to refresh dynamic storage. The processors are small enough that we expect to fit four of them with 1024 nibbles of storage each on a chip with  $\lambda=1$  micron, or one per chip with  $\lambda=2$  micron. By adhering to the principles of smallness and regularity, the processors achieve very good duty-factors on their internal parts, and are expected to be very fast.

#### 5. Mapping TMPL Primitives onto the Hardware

The hardware design places a pair of queues between the two communicating processors. It supports three operations: putting a message at the end of the output queue, removing the top element of the input queue, and non-destructively examining the top element of the input queue. These operations correspond to imperative output and conditional input. TMPL programs also contain imperative inputs and conditional outputs, and these must be transformed into the two forms

that are implemented. We discuss that mapping here.

### 5.1. Imperative Input

An imperative input statement is semantically identical to a conditional statement with only one guard, the expression form of the imperative, and a no-op action following the guard;

$$p?msg \equiv [ p?msg \rightarrow \text{SKIP} ]$$

Message expressions remain in the **maybe** state until there is some activity on the port named in the expression. As long as not all of the guards in a conditional or loop statement are false, the statement will not terminate. In this case, the processor will wait until something arrives at the port. If the message matches, the input is done, the no-op is executed, and the conditional statement is completed. If some other message is pending on the port, the conditional statement aborts, as does a mismatched imperative.

We have just changed an imperative input into a conditional, and in the process, may have created an illegal conditional to conditional communication. If we stopped here, that would be a major concern. However, we are also going to remove all conditional outputs in this compilation step. With only conditional inputs and imperative outputs remaining, it is impossible to have an invalid pairing of conditionals.

### 5.2. Conditional Output

When conditional output is used, the processor is asking the port whether the other processor is waiting to input the message. The conditional output can be restated as a pair of communications. First the processor issues a *message statement* asking, in essence, if the other processor wants the message. This imperative communication goes *outside* the body of the conditional or loop statement that contains the original conditional output. That expression is replaced with a *conditional input* waiting for a positive reply to the query. The replacement technique is shown below.

$$[ p!msg \rightarrow \text{command} ] \quad \text{becomes} \quad p!query ; [ p?yes \rightarrow p!msg; \text{command} ]$$

Notice that a new message, *query*, has been introduced. The program in the processor at the other end of the port must now be changed to accept the new message. The message typing discussed earlier makes it trivial to identify the imperative input that is paired with the original conditional output, and replace it with a conditional statement that waits for the *query* message, answers it, and then inputs *msg*. The replacement code is given below.

$$p?msg; \quad \text{becomes} \quad [ p?query \rightarrow p!yes ; [ p?msg \rightarrow \text{SKIP} ] ]$$

The example above uses a conditional statement with a single guard. The same replacement technique is used for statements with multiple guards, but there is a subtle difference in the code in the processor doing the imperative inputs: it must clear *all* of the query messages before requesting the original message. Figure 6 shows a loop statement with three conditional outputs as guards, and an imperative input that matches one of them. We show how the code in both processors is rewritten to remove the conditional outputs from one and the imperative input from the other.

Removing conditional outputs from the source code is not as clean as one might like. The connection plan for the tree must be used to identify the port connections, since both processors involved in the conditional output must be modified. A solution that requires changing only the program in the processor actually doing the conditional output is preferred, but elusive.

## 6. Conclusions

We have described a strategy for providing local communication among small, simple processors connected like a binary tree. The design has both software and hardware components.

The software, a compiler, allows the programmer to write programs for trees with arbitrary fanout using a rich set of communication primitives. The compiler recursively applies a mapping algorithm that assigns logical ports to physical ones until the tree with arbitrary fanout has been



Original Source	Rewritten Source
$\begin{array}{l} \{ p!m1 \rightarrow SKIP \\   p!m2 \rightarrow SKIP \\   p!m3 \rightarrow SKIP \\ \} \end{array}$	$\begin{array}{l} p!query1; p!query2; p!query3; \\ \{ p?ack1 \rightarrow p!m1; SKIP \\   p?ack2 \rightarrow p!m2; SKIP \\   p?ack3 \rightarrow p!m3; SKIP \\ \} \end{array}$
$p?m2;$	$\begin{array}{l} [ p?query1 \rightarrow SKIP ]; \\ [ p?query2 \rightarrow \\ \quad [ p?query3 \rightarrow SKIP ]; \\ \quad p!ack2; [ p?m2 \rightarrow SKIP ] \\ ]; \end{array}$

Figure 6. Removing Conditional Outputs

made binary. Skeleton programs for the intermediate processors that pass messages between parent and child are produced.

The hardware design places a pair of queues between processors. The queues implement a subset of the software primitives; the others can be mapped onto them by the compiler. The advantage of the queue scheme is its simplicity. Because the power of the tree machine comes from the *number* of processors, and not the capabilities of an individual one, a hardware design that is small, simply described, and easy to use, is desired.

**Acknowledgements.** The research described here was sponsored in part by the Defense Advanced Research Projects Agency, ARPA order #3771, and monitored by the Office of Naval Research under contract #N00014-79-C-0597. Our starting point was a sketch of the machine presented in S. A. Browning's doctoral dissertation.<sup>1</sup> A group of nine people met regularly to refine that sketch into a first implementation in silicon. Martin Rem, Lennart Johnsson, and Peggy Li made valuable contributions to the discussions of mapping problems between the notational abstraction and the machine implementation. Each potential design was examined in light of the requirements of the notation until one was found that satisfied both the programmers and the hardware designers. A compiler like the one described here is being implemented by S. A. Browning at Bell Laboratories.

A tree machine processor has been designed and is being laid out by C. L. Seitz, Howard Derby, Chris Kingsley, and Chris Lutz. Erik deBenedictis and Peggy Li have written a collection of programs to evaluate the processor design.

## References

1. Sally A. Browning, *The Tree Machine: A Highly Concurrent Computing Environment*, California Institute of Technology (1980). Computer Science Technical Report #3760
2. E. W. Dijkstra, *A Discipline of Programming*, Prentice-Hall, Englewood Cliffs, New Jersey (1976).
3. C. A. R. Hoare, "Communicating Sequential Processes," *C.ACM* **21**(8), pp. 666-677 (August, 1978).
4. Charles L. Seitz, "Self-Timed VLSI Systems," *Proc. Caltech Conference on Very Large Scale Integration*, pp. 345-356 (January, 1979).
5. Ivan E. Sutherland and Carver A. Mead, "Microelectronics and Computer Science," *Scientific American* **237**(3), pp. 210-228 (September, 1977).

## Appendix 1. Templates for Padding Programs

### 1. Imperative output: $c(i)!\text{msg}$

```

parent:
  [ odd(i) → l!msg((n-3)/2,(i+1)/2)
  | even(i) →
    [ n=3 → r!msg
    | n>3 → r!msg((n-4)/2,i/2)
    ]
  ]

padding:
  p?msg(d,p) →
    [ d=0 AND odd(p) → l!msg
    | d<2 AND even(p) → r!msg
    | d>0 AND odd(p) → l!msg((d-1)/2,(p+1)/2)
    | d>1 AND even(p) → r!msg((d-2)/2,p/2)
    ]

```

### 2. Imperative Broadcast Output: $c(*)!\text{msg}$

```

parent:  l,r!msg

padding:  p?msg → l,r!msg

```

### 3. Imperative Input: $c(i)?\text{msg}$

```

parent:
  [ odd(i) →
    l!request((n-3)/2,(i+1)/2) ;
    l?msg
  | even(i) →
    [ n=3 → SKIP
    | n>3 → r!request((n-4)/2,i/2)
    ] ;
    r?msg
  ]

padding:
  p?msg(d,p) →
    [ d=0 AND odd(p) → l?msg
    | d<2 AND even(p) → r?msg
    | d>0 AND odd(p) →
      l!request((d-1)/2,(p+1)/2) ;
      l?msg
    | d>1 AND even(p) →
      r!request((d-2)/2,p/2) ;
      r?msg
    ] ;
  p!msg

```



**4. Imperative Broadcast Input:  $c(*)?msg$** 

parent:

```

    l!req((n-3)/2);
    [ l?msg → SKIP
      | l?no →
        [ n=3 → SKIP
          | n>3 → r!req((n-4)/2)
        ] ;
      r?msg
    ]

```

padding:

```

    BOOLEAN got;

```

```

    p?req(d) →
      got:=FALSE ;
      [ d=0 →
        { NOT got AND l?msg → got:=TRUE }
      | d>0 →
        l!req((d-1)/2);
        [ l?msg → got:=TRUE
          | l?no → SKIP
        ]
      ] ;
      [ NOT got AND d<2 →
        { NOT got AND r?msg → got:=TRUE }
      | NOT got AND d>1 →
        r!req((d-2)/2);
        [ r?msg → got:=TRUE
          | r?no → SKIP
        ]
      ] ;
      [ got → p!msg
        | NOT got → p!no
      ]

```

### 5. Conditional Output: $c(i)!\text{msg} \rightarrow$

parent:

```
[ odd(i) → !msg((n-3)/2,(i+1)/2)
| even(i) →
  [ n=3 → SKIP
  | r>3 → r!msg((n-4)/2,i/2)
  ]
]
```

l,r?yes →

```
whatever was here
[ odd(i) → !msg((n-3)/2,(i+1)/2)
| even(i) →
  [ n=3 → SKIP
  | r>3 → r!msg((n-4)/2,i/2)
  ]
]
```

padding:

BOOLEAN got;

```
| p?msg(d,p) →
  [ d=0 AND odd(p) →
    got := FALSE ;
    { NOT got AND !msg → got:=TRUE };
    [ got → p!yes
    | NOT got → SKIP
    ]
  | d<2 AND even (p) →
    got := FALSE ;
    { NOT got AND r!msg → got:=TRUE };
    [ got → p!yes
    | NOT got → SKIP
    ]
  | d>0 AND odd(p) → !msg((d-1)/2,(p+1)/2)
  | d>1 AND even(p) → r!msg((d-2)/2,p/2)
  ]
| l,r!yes → p!yes
```

**6. Conditional Input:  $c(i)?msg \rightarrow$** 

parent:

```

[ odd(i)  $\rightarrow$  l?req((n-3)/2,(i+1)/2)
| even(i)  $\rightarrow$ 
  [ n=3  $\rightarrow$  SKIP
  | r>3  $\rightarrow$  r!msg((n-4)/2,i/2)
  ]
]

l,r?msg  $\rightarrow$ 
  whatever was there
  [ odd(i)  $\rightarrow$  l?req((n-3)/2,(i+1)/2)
  | even(i)  $\rightarrow$ 
    [ n=3  $\rightarrow$  SKIP
    | r>3  $\rightarrow$  r!msg((n-4)/2,i/2)
    ]
  ]

```

padding:

```

BOOLEAN got;

| p?req(d,p)  $\rightarrow$ 
  [ d=0 AND odd(p)  $\rightarrow$ 
    got:=FALSE ;
    { NOT got AND l?msg  $\rightarrow$  got:=TRUE } ;
    [ got  $\rightarrow$  p!msg
    | NOT got  $\rightarrow$  SKIP
    ]
  | d<2 AND even(p)  $\rightarrow$ 
    got:=FALSE ;
    { NOT got AND r?msg  $\rightarrow$  got:=TRUE } ;
    [ got  $\rightarrow$  p!msg
    | NOT got  $\rightarrow$  SKIP
    ]
  | d>0 AND odd(p)  $\rightarrow$  l!req((d-1)/2,(p+1)/2)
  | d>1 AND even(p)  $\rightarrow$  r!req((d-2)/2,i/2)
  ]
| l,r?msg  $\rightarrow$  p!msg

```

## 7. Conditional Broadcast Input: $c(*)?msg \rightarrow$

parent:

```

l!req((n-3)/2) ;
[ l?yes → SKIP
  | l?no →
    [ n=3 → SKIP
      | n>3 → r!req((n-4)/2)
    ] ;
  [n=3 → SKIP
    | n>3 AND r?yes → SKIP
    | n>3 AND r?no → SKIP
  ]
]

```

l,r?msg →

```

whatever was there
l!req((n-3)/2) ;
[ l?yes → SKIP
  | l?no →
    [ n=3 → SKIP
      | n>3 → r!req((n-4)/2)
    ] ;
    [ r?yes → SKIP
      | r?no → SKIP
    ]
  ]
]

```

padding:

BOOLEAN got;

p?req(d) →

```

got:=FALSE ;
[ d=0 →
  { NOT got AND l?msg → got:=TRUE }
  | d>0 →
    l!req((d-1)/2);
    [l?yes → got:=TRUE ; l?msg
      | l?no → SKIP
    ]
  ] ;
[ NOT got AND d<2 →
  { NOT got AND r?msg → got:= TRUE }
  | NOT got AND d>1 →
    r?req((d-2)/2);
    [r?yes → got:=TRUE ; r?msg
      | r?no → SKIP
    ]
  ] ;
[got → p!yes ; p!msg
  | NOT got → p!no
]

```

## THE TORUS: AN EXERCISE IN CONSTRUCTING A PROCESSING SURFACE

Alain J. Martin  
Philips Research Laboratories  
5600 MD Eindhoven  
The Netherlands

Abstract. A "Processing Surface" is defined as a large, dense, and regular arrangement of processor and storage modules on a two-dimensional surface, e.g. a VLSI chip. A general method is described for distributing parallel recursive computations over such a surface. Scope rules enforcing the "locality" of variables and procedure parameters are introduced in the programming language. These rules and a particular interconnection of the modules on the surface make it possible to transmit parameter and variable values between modules without using extraneous communication actions.

The choice of the Processing Surface topology for binary recursive computations is discussed and a torus-like topology is chosen.

### 0. INTRODUCTION

Let us call a "Processing Surface" a large, dense, regular arrangement of processor and storage modules on a two-dimensional surface, e.g. a VLSI chip. How can a computation be distributed over such a surface? What are the arrangements of the modules on the surface best suited for a certain class of computations?

We propose to explore this problem in the following direction. In such an environment, an action on a variable differs in complexity (in terms of the number of elementary steps necessary to perform the action) depending on the distance between the processor module performing the action and the storage module containing the variable. We want to reflect this issue at the programming level by introducing scope rules defining the distance between the program component where a variable is declared, and the program components where the variable can be used.

Since we expect intense communications between the program components, we expect assignments of the form  $x:=y$  where  $x$  and  $y$  belong to two adjacent components (this assignment can take the form of a procedure call or a pair of matching communication actions) to occur as frequently as assignments between variables of the same component. In most distributed systems, the first type of assignment is an order of magnitude more complex than the second one. We consider this hidden discrepancy between equivalent actions unacceptable. We will show that it is possible to define some locality rule for the program variables, and to organize the processor and storage modules on the surface such that no discrepancy of this sort appears. In such a case, the Processing Surface is said to be "continuous".

Furthermore, since for instance inverting a  $2 \times 2$  matrix does not require as much parallelism as inverting a  $1000 \times 1000$  matrix, the potential parallelism of an algorithm should not be fixed beforehand (e.g. by the number of available processors) but should be determined dynamically according to the needs of the computation. The component actions of a computation should be created and destroyed as the computation proceeds, and should be automatically distributed over the available modules.

## 1. THE GENERAL METHOD

The general method we use has been described in [1]. We shall recall it briefly.

The component actions of a computation - the "nodes" - are regarded as the vertices of a graph - the "computation graph" - which grows and shrinks during the computation. An edge - a "channel" - between two nodes means that one of the two, say node A, has created the other, say node B, by a procedure call, and that A and B communicate directly with each other. A is the "father" of B, and B is a "son" of A. Thanks to a parallel procedure call, a father may create several sons simultaneously. The father/son relation defines a partial ordering of the nodes, and all nodes that are not relatively ordered can be performed in parallel.

A computation graph grows and shrinks through a given finite "implementation graph", whose vertices - the "cells" - represent the available modules, and the edges - the "links" - the communication possibilities between modules. Each node is mapped on a cell, and each channel on a link.

Hence, each cell may have to accommodate an unbounded number of nodes. Since a cell represents a very small number of sequential automata (in most cases, one!), the activities of all nodes simultaneously present in a cell have to be sequentialized in some way. But such a sequentialization may introduce deadlock. The main result of [1] is to prove that the nodes of a cell can be interleaved without introducing deadlock provided that the grain of interleaving be correctly chosen. The solution is very simple in that it does not require any particular knowledge about the nodes or the implementation graph nor complicated scheduling.

In this paper we shall consider a special class of computations, namely recursive computations. For this class of computations we shall describe how to implement a continuous Processing Surface, and we shall propose a torus-like topology for the implementation graph.

## 2. RECURSION

Much has been said about the use of recursion for parallel programming. The reader is referred to the abundant literature on this subject. For the sake of simplicity, we shall restrict ourselves to one of the most usual recursive methods, namely "divide-and-conquer" (also called "recursive doubling"). Divide-and-conquer algorithms are particularly interesting in that they produce binary trees as computation graphs. Binary trees are regular structures and each node has an outdegree of two, which is interesting in view of their mapping onto a two-dimensional surface.

Parallelism is introduced only by calling two procedures "in parallel". The possibility of further increasing parallelism by pipelining the parameters will not be mentioned although it can easily be added. Nevertheless this class of algorithms is large enough (in particular numerical algorithms) for the exercise to be realistic.

### 3. THE LOCALITY OF VARIABLES AND PARAMETERS

Since a node is created by a procedure call, a node is a procedure instance with its own program counter, and its set of variables and parameters. The following rules define the "locality" of variables and parameters.

- . The unit of locality is the node: a variable declared inside a node is local to that node.
- . A variable local to a node A is a neighbour for all son nodes of A .

Since the father/son relation between nodes is not transitive, the locality or neighbourhood of a variable with respect to a node is not transitive either: if a node P1 calls a node P2 which calls a node P3 , a variable local to P1 is neighbour for P2 , but not for P3 .

Three types of parameters are used:

- . An input parameter is used to "import" a parameter value into a son node, by an assignment of the actual parameter value to the formal parameter variable.
- . An output parameter is used to "export" a value from a son node to its father by an assignment of the formal parameter value to the actual parameter variable.
- . A reference parameter is used both to import and to export, but by a process of substitution, or "aliasing": the formal parameter replaces the actual parameter in the son node (it is another name for the same variable).

In the case of the input and the output parameters, the formal parameter is local to the son.

In the case of the reference parameter, the formal parameter has the same locality as the actual parameter. The formal parameter is thus not local to the son.

Assume that the value x of a variable is to be imported from a father node P1 into a son node P2 . Either an input or a reference mechanism can be used. Assume now that x is to be passed again from P2 to a son node P3 . If x was passed from P1 to P2 as an input parameter, x will be local to P3 if it is passed as an input parameter from P2 to P3 , and neighbour to P3 if it is passed as a reference parameter from P2 to P3 . But if x was passed from P1 to P2 as a reference parameter, x will neither be local nor neighbour to P3 , whether it be passed as an input or as a reference parameter from P2 to P3 .

(In the case where a value is to be exported from a son node to a father node, exactly the same differences hold according to whether it is passed as an output or a reference parameter.)

Hence, the locality or neighbourhood of a reference parameter with respect to a node is not transitive whereas that of an input or output parameter is. But when a value  $x$  is passed as an input or an output parameter from node  $P$  to node  $Q$ , by definition  $x$  is copied from the storage area of  $P$  into the storage area of  $Q$ . No copying is necessary when  $x$  is passed as a reference parameter.

The repetitive transport of values via global variables and reference parameters could be used in its full generality, but we propose to restrict its use by the following "locality rule".

Locality rule: An action of a node involves only variables and parameters that are local and/or neighbour for the node.

(Whether global variables should be used at all is doubtful. They have been included for the sake of completeness.) We shall see that this locality rule permits the implementation of a continuous Processing Surface.

#### 4. IMPLEMENTATION OF A CONTINUOUS PROCESSING SURFACE

Definition: A Processing Surface is said to be "continuous" when any action performed on the surface involves only variables that are directly accessible to the processor performing the action, i.e. accessible by elementary read or write operations.

Hence, if we succeed in implementing a continuous surface, we shall have suppressed any form of extraneous communication action for accessing variables.

According to the general method, we know that if node  $N1$  is mapped on cell  $C1$ , a son node  $N2$  of  $N1$  is mapped on a neighbour cell  $C2$  of  $C1$ . For node  $N1$  to be mapped on  $C1$  means that the local variables and parameters of  $N1$  must be allocated in the storage module associated with  $C1$ , and the same for  $N2$  relative to  $C2$ . Let  $M1$  and  $M2$  be the storage modules associated with  $C1$  and  $C2$ , respectively. According to the locality rule, any action of  $N2$  may involve variables located in  $M1$  and  $M2$ . The set  $\{M1, M2\}$  is called the "locality area" of  $N2$ . In the case where the computation graph is a tree, the locality area of a node consists of at most two elements.

As a direct consequence of the locality rule and of the definition of a continuous Processing Surface, the Processing Surface is continuous if the property  $C(N)$  holds for any node  $N$ .

$C(N)$  : any action of  $N$  is performed by a processor directly connected to the two storage modules of the locality area of  $N$ .

We shall describe a strategy for placing the processor and storage modules on the implementation graph, and for distributing the actions and the variables of the nodes over the processor and storage modules, such that  $C(N)$  holds for any node  $N$ .

This strategy thus implements a continuous Processing Surface.

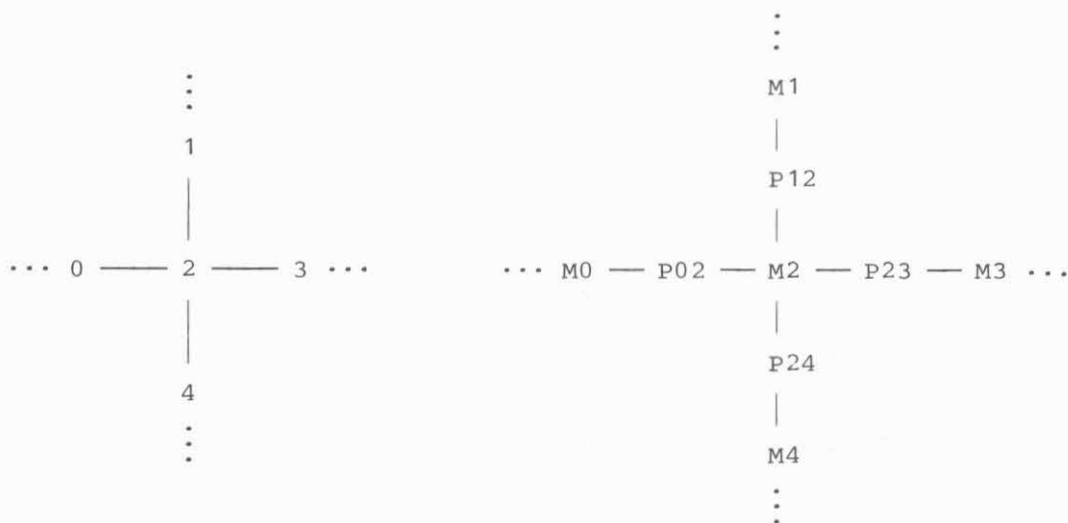


1) The placing strategy is directly suggested by the property  $C(N)$  .

A storage module is placed at each vertex, and a processor module at each edge of the implementation graph.

(See an example on fig. 1.) Hence, each processor has direct access to two storage modules, and each storage module is shared by as many processors as the degree of the vertex where it is placed.

2) Assume that  $C(F)$  holds for a node  $F$  . For instance,  $F$  has been created in cell 2 of fig. 1(a); its local variables are in  $M2$  , its neighbour variables in  $M1$  , and its actions are processed by  $P12$  (see fig. 2).



(a) implementation graph

(b) processor and storage placement

Fig. 1.

Assume that at some stage in the computation of  $F$  two son nodes  $R$  and  $D$  (for right and down) of  $F$  are to be created in cells 3 and 4, respectively. The locality areas of  $R$  and  $D$  must then be  $(M2, M3)$  and  $(M2, M4)$  , respectively (see fig. 2).

This means that  $C(R)$  and  $C(D)$  will hold if and only if  $R$  and  $D$  are processed by  $P23$  and  $P24$  , respectively. Upon reaching the procedure calls of  $R$  and  $D$  in the procedure body of  $F$  ,  $P12$  must transmit the creation of  $R$  and  $D$  to  $P23$  and  $P24$  .

Since, by construction  $P12$  ,  $P23$  , and  $P24$  share a common store, namely  $M2$  , the transmission of procedure calls is a simple and local action:  $P12$  adds the names of  $R$  and  $D$  to the lists - located in  $M2$  - of nodes to be processed by  $P23$  and  $P24$  , respectively.

A processor switches from one node to the other upon a procedure call in the same way as in a multiprogramming system a processor switches from one process to another upon a P-operation on a zero semaphore. We shall not describe the implementation in more detail.

Hence, if  $C(F)$  holds for a node  $F$ ,  $C(R)$  and  $C(D)$  hold for the two son nodes of  $F$ . Observe that the above strategy is independent of the topologies of the implementation graph and of the computation tree. The root node  $P$  of the computation tree is created by the "environment" of the computation. At least one cell of the implementation graph - a root cell - is connected to the environment. It is easy to map  $P$  onto a root cell in such a way that  $C(P)$  holds.

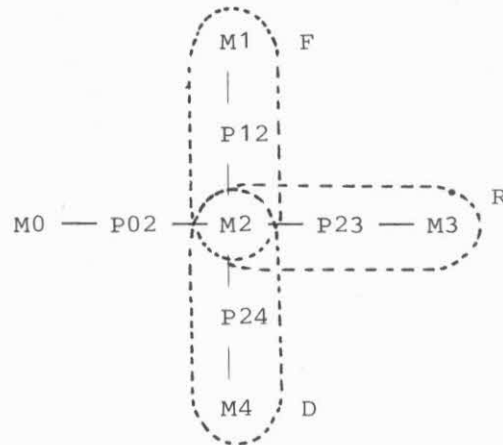


Fig. 2. locality areas

## 5. THE CHOICE OF THE IMPLEMENTATION GRAPH

We look for a finite implementation graph such that 1) an arbitrary binary tree can be mapped onto it without knowing the sizes of the tree and of the graph, 2) the nodes of the tree are optimally spread over the cells of the graph.

Because of 1), we aim at "simulating" an infinite graph on a finite one. Let us assume that we could indeed construct an infinite implementation graph, which graph would we choose? Since we are looking for graphs that can be represented in the plane by regular and dense structures, we are bound to choose between the three regular tessellations of the plane, which are the square, the triangular, and the hexagonal tessellations. (Although the infinite binary tree is regular, it is not dense, because it grows exponentially and therefore cannot be represented with minimal constant edge lengths.)

We have chosen the square tessellation, although the hexagonal is also interesting. We shall first discuss the problems of mapping a binary tree onto an infinite grid. We shall then simulate the infinite grid on a finite grid.

## 6. THE INFINITE GRID AS AN IMPLEMENTATION GRAPH

An infinite grid is a graph such that: for  $i \geq 0$  and  $j \geq 0$ , vertex  $(i, j)$  is connected with vertex  $(i+1, j)$  and vertex  $(i, j+1)$ .

The mapping of a binary tree on the grid is obvious. The root of the tree is mapped onto vertex  $(0, 0)$ . If a node is mapped on vertex  $(i, j)$ , then its right son  $R$  is mapped on vertex  $(i, j+1)$ , and its down son  $D$

is mapped on vertex  $(i+1, j)$ . When an exponential structure (the binary tree) is mapped on a quadratic one (the grid) a congestion problem is created: vertex  $(i, j)$  of the grid may have to accommodate up to  $(i+j)!/i!*j!$  nodes of the binary tree simultaneously.

## 7. THE STRAIGHT TORUS

The problem now is to simulate the infinite grid on a finite one. For reasons of symmetry we choose a square grid of  $M \times M$  cells. (We shall return to this choice later.) The first solution consists in connecting cell  $(x, y)$  of the finite grid  $(0 \leq x, y < M)$  to the cells:

$(x, (y+1) \bmod M)$   
and  $((x+1) \bmod M, y)$ .

This amounts to connecting with each other the corresponding elements of the first and last columns, and those of the first and last rows. The volume obtained is topologically similar to a torus.

Consider an arbitrary cell  $(i, j)$  of the infinite grid and the cell  $(x, y)$  of the finite grid on which it is mapped. According to the above connecting rule, we have:

$$\begin{aligned} i &= x + k \times M \\ j &= y + l \times M \end{aligned} \quad (R)$$

This relation describes the tiling of the infinite grid by square tiles of size  $M \times M$ : if  $(i, j)$  are the coordinates of a cell of the infinite grid, then  $(x, y)$  are its coordinates in the tile  $(k, l)$  (see fig. 3.).

The congestion problem can be solved in the following way. Consider the infinite grid. When a vertex is occupied by a node  $N$  of the computation tree, no other node is accepted by the vertex until  $N$  and the subtree attached to  $N$  have terminated their activity. It is easy to prove that this cannot lead to deadlock on the infinite grid. But this solution cannot be used in a straightforward manner for the torus without danger of deadlock. Assume that a cell of the torus is occupied by the node  $N_1$ , and a new node  $N_2$  is not accepted by the cell. It may occur that  $N_2$  belongs to the subtree of  $N_1$ . This would be a deadlock. For each node of the computation tree, it is recorded to which tile the node belongs. When a cell is occupied by a node  $N_1$ , it may refuse a node  $N_2$  only if  $N_2$  belongs to the same tile as  $N_1$ . (If two nodes belong to the same tile, it is impossible that one belongs to the subtree of the other.)

## 8. THE PROPAGATION PATTERN

Assume that all cells and all nodes in the cells have similar behaviours, and that the propagation speeds are similar in all directions even in the case of an asynchronous implementation. Then we can say that in a phase of homogeneous expansion or contraction of the computation, there is a front wave of active nodes which are located at a maximum distance from the root, i.e. on a diagonal  $i + j = K$  of the infinite grid, which we shall call the "active diagonal".

At step  $K$  of the computation, the complete computation tree contains  $2 \times (K-1)$  active nodes (the leaves). But at step  $K$  of the computation, at most  $K(K+1)/2$  cells of the infinite grid can be active, and if the strategy for reducing congestion is applied, at most  $K$ : the cells of the active diagonal. As a consequence, the  $2 \times (K-1)$  leaf nodes cannot be active

simultaneously; their activities have to be sequentialized. The hypothesis of homogeneous expansion and contraction then does not strictly hold anymore because not all cells on a diagonal have to accommodate the same number of leaf nodes, and therefore the contraction of the computation will not start in all cells at the same time. But it is an acceptable approximation.

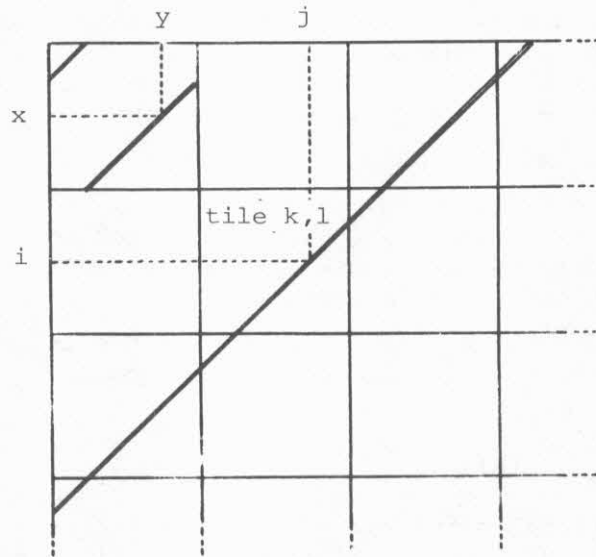


Fig. 3.

Fig. 3 shows that the active diagonal of the infinite grid is mapped on at most two diagonals of the finite grid, i.e. at most  $M$  cells out of the  $M \times M$  are active.

(Algebraically, for a given value of  $i + j$ , there are at most two values of  $x + y$  ( $0 \leq x + y < 2M - 1$ ) fulfilling  $R$ , namely:

$$(i + j) \bmod M \\ \text{if } (i + j) \bmod M < M - 1 : (i + j) \bmod M + M .$$

Hence, if the active diagonal approximation is correct, the straight torus topology leads to a poor distribution of the computation over the Processing Surface.

## 9. THE TWISTED TORUS

Obviously, the drawback of the straight torus is caused by the symmetry of the tiling of fig. 3 around the axis  $i = j$ . We can destroy the symmetry by shifting the tiling by one position and in one direction, as shown by fig. 4. Now we see that for the same active diagonal, more diagonals of the finite grid are occupied. In fact, it can be proved that the distribution of the active diagonal over the finite grid is now optimal: if the active diagonal contains no more than  $M \times M$  nodes, no two nodes of the active diagonal are mapped on the same cell of the torus.

This tiling corresponds to the tiling relation:

$$i = x + k \times M - 1 \\ j = y + l \times M .$$

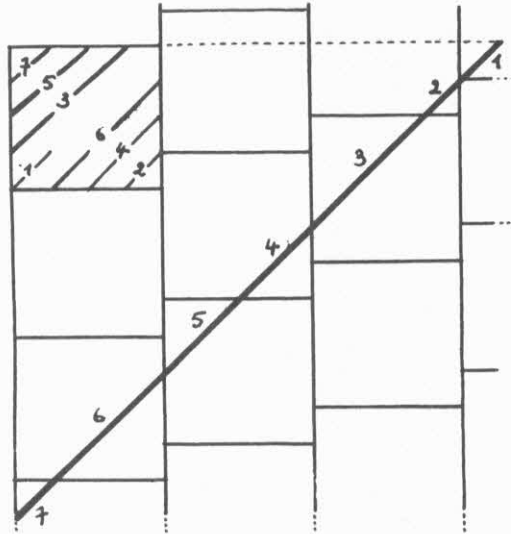


Fig. 4.

#### 10. THE DOUBLY TWISTED TORUS

The same result could have been reached by using a rectangular straight torus of  $M \times P$  cells where  $M$  and  $P$  are relative primes. The difference is that in a twisted torus, a horizontal chain of nodes, i.e. the succession of nodes with constant  $i$ , is mapped on a cycle containing all cells of the torus, i.e. on a cycle of length  $M \times M$ . On the rectangular torus, such a structure is mapped on a cycle of length  $M$  (one row of the torus). In both cases a vertical chain (constant  $j$ ) is mapped on the cells of only one column.

In view of certain degenerate binary trees, which reduce to a chain of only right or left procedure calls, it could be interesting to twist the torus in both directions in such a way that a vertical chain is also mapped on all cells of the torus.

To avoid reintroduction of the symmetry, the torus must be twisted in opposite directions in the two dimensions (e.g.  $+1$  for the rows, and  $-1$  for the columns).

The fact that the corresponding tiling relation:

$$i = x + k \times M - 1$$

$$j = y + l \times M + k$$

has no solution for  $(i, j) = (M(q+1) - p, pM + q)$  means that such a tiling does not represent a "plane" surface. We mean that if, on the infinite grid, point  $B$  is reached from point  $A$  by  $r$  horizontal steps and  $s$  vertical ones,  $B$  is also reached from  $A$  by any permutation of these steps. This is no longer true for this doubly twisted torus.

This is shown by the following counter-example. Consider the  $3 \times 3$  doubly twisted torus of fig. 5(a). From point  $A$ , one horizontal step (indicated in fig. 5 by a dotted path), followed by one vertical step (indicated in fig. 5 by a dashed path) leads to point  $B$  (fig. 5(b)). From point  $A$ ,

one vertical step followed by one horizontal step leads to point C (fig. 5(c)). Points B and C are different.

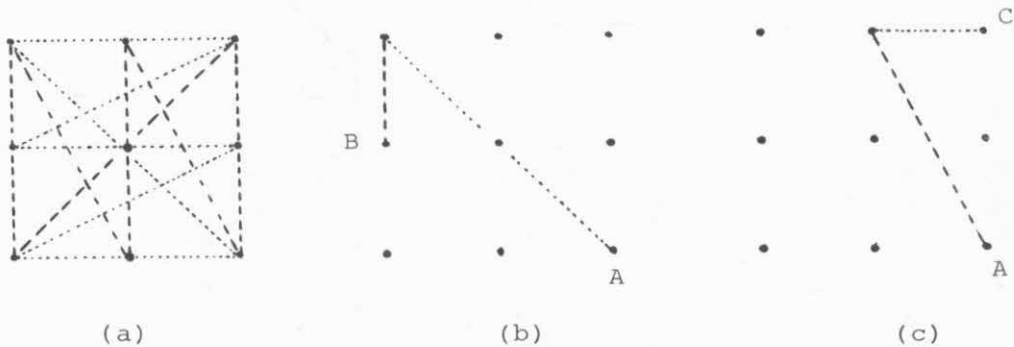


Fig. 5.

This drawback is only significant if one wants to implement computation graphs other than trees. In a tree, there is only one path between two points. If one wants to maintain the planarity of the torus, one must look for tessellations of the plane that are not square, and yet still use the double twist. Two are given in fig. 6. The first one is due to Carlo Sequin [2].

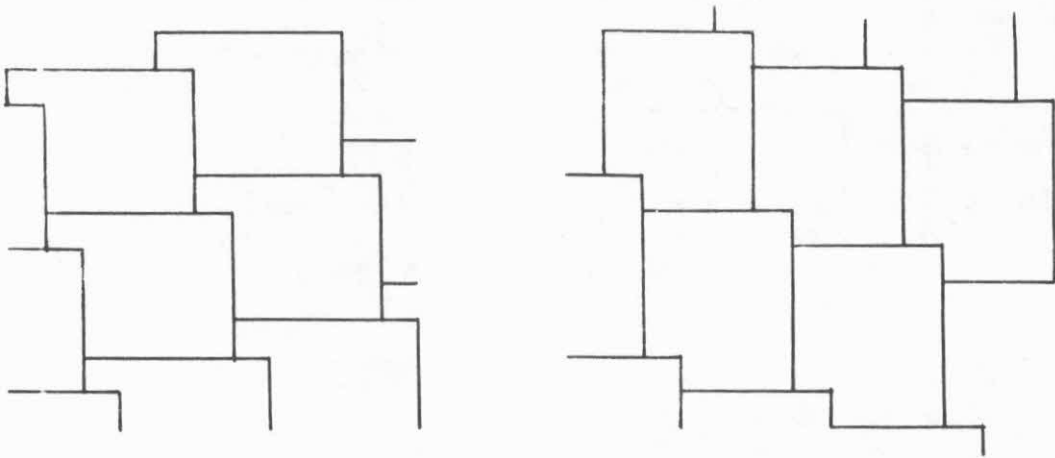


Fig. 6.

## 11. CONCLUSION

A method has been proposed to construct highly parallel and distributed systems where the basic hardware building blocks are whole processor and storage modules, and the basic software building block is the procedure. The main aspects of the method are the following.

First, on such a Processing Surface the location of variables relative to the processors using them is a relevant factor. Scope rules have therefore been introduced in the programming language, which allow the programmer to determine the "distance" between the variables or procedure parameters and the actions where they are used.

Second, since intense communications between adjacent modules are expected, we have attempted to smooth away the discontinuity in variable access caused by the boundary between storage modules. For this purpose, the access to distant variables has been limited to neighbour variables by a "locality rule". Furthermore the processor and storage modules have been arranged in such a way that no extraneous communication procedure is needed to "move" variable values over a storage module boundary. The result is called a continuous Processing Surface.

Third, by using a "boundary-less" topology for the surface (here, a torus), the automatic diffusion of a divide-and-conquer computation through the surface leads to an optimal spreading of the load over the modules. The programmer need not know the actual number of modules, and no complicated scheduling is required.

## 12. HISTORY AND ACKNOWLEDGEMENTS

The first torus machine was built at the beginning of 1979 at Philips Research Laboratories. It is a twisted torus of 36 cells. Each cell consists of two INTEL chips (one processor with a 1K byte ROM, and one 256 byte RAM.). It is not a Processing Surface but a network of machines communicating by explicit message exchanges.

Acknowledgement is due to W.J. Lippmann and G.A. Slavenburg for their invaluable cooperation during the construction of this machine. Without their hardware and software competence, it would never have been completed within such a short term. The fact that it was completed within 3 months is also a consequence of the regularity of the structure. Acknowledgement is also due to C.S. Scholten for several valuable comments on the first paper on the subject [3], and to Alan Davis whose comments on the manuscript led to many improvements.

## REFERENCES

- [1] A.J. Martin: "A Distributed Implementation Method for Parallel Programming." Proceedings IFIP congress 80 - October 1980.
- [2] C.H. Sequin: "Doubly Twisted Torus Networks for VLSI Processor Arrays." December 3, 1980 - draft.
- [3] A.J. Martin: "A Distributed Architecture for Parallel Recursive Computations." Philips. AJM18 - September 1979.





Architecture for VLSI Design of  
Reed-Solomon Encoders\*

K.Y. Liu

Jet Propulsion Laboratory, Pasadena, CA. 91109

Abstract

In this paper, the logic structure of a universal VLSI chip called the symbol-slice Reed-Solomon (RS) encoder chip is presented. An RS encoder can be constructed by cascading and properly interconnecting a group of such VLSI chips. As a design example, it is shown that a (255,223) RS encoder requiring around 40 discrete CMOS IC's may be replaced by an RS encoder consisting of four identical interconnected VLSI RS encoder chips. Besides the size advantage, the VLSI RS encoder also has the potential advantages of requiring less power and having a higher reliability.

I. Introduction

Reed-Solomon (RS) codes [1] are non-binary BCH codes. These codes can correct both random and burst errors over a communication channel. Recently concatenated coding systems using RS codes as the outer codes have been proposed for space communication to achieve very low error probabilities [2]-[7]. Several deep space flight projects such as the Voyager at Uranus encounter, the Galileo, and the International Solar Polar Mission (ISPM) have also considered using the concatenated RS/Viterbi channel coding scheme. Hence RS codes are quite important for space communications.

The complexity of an RS encoder is proportional to the error-correcting capability of the code, the speed of the encoding, and the interleaving level used [4]. For reliable space communication there is a need to use RS codes

---

\*This paper presents one phase of research conducted at the Jet Propulsion Laboratory, California Institute of Technology under contract No. NAS-7-100 sponsored by the National Aeronautics and Space Administration.

with large error-correcting capability and large interleaving level [4], [5], [8]. Hence one is especially interested in minimizing the complexity of RS encoders for space communication applications. In a spacecraft the power, size, and reliability requirements are usually quite severe. Thus there is considerable interest in a VLSI (Very Large Scale Integration) RS encoder which has the potential for significant savings in size, weight, and power while at the same time providing higher reliability over an RS encoder implemented in discrete logic circuits.

This paper introduces a symbol-sliced logic structure suitable for a VLSI implementation of RS encoders. By cascading and properly interconnecting a group of such VLSI chips, each consisting of a fixed portion of the encoder, it is possible to obtain an RS encoder with any desired error-correcting capability and interleaving level. As a design example, it is shown that a (255,223) RS encoder requiring 40 discrete CMOS IC's may be replaced by an RS encoder consisting of four identical interconnected VLSI encoder chips.

## II. Reed-Solomon Encoding Procedures

An RS codeword has  $(2^J - 1)$  symbols, where each symbol has  $J$  bits. Of the  $(2^J - 1)$  symbols there are  $(2^J - 1 - 2E)$  information symbols and  $2E$  parity-check symbols, where  $E$  is the number of symbols an RS code is able to correct. If one treats the  $(2^J - 1 - 2E)$  information symbols as the coefficients of the polynomial

$$f(x) = x^{2E} \left( s_{2^J-1-2E} + s_{2^J-2-2E} x + \dots + s_2 x^{2^J-2-2E} + s_1 x^{2^J-1-2E} \right)$$

where  $s_i$  is the  $i$ th transmitted symbol, then the  $2E$  parity-check symbols can be obtained as the coefficients of the remainder of

$$f(x)/g(x)$$

where  $g(x)$  is the generator polynomial [9] of the code. Usually  $g(x)$  is defined as

$$g(x) = \prod_{i=1}^{2E} (x - \alpha^i) = \sum_{j=0}^{2E} g_j x^j$$

where  $\alpha$  is a primitive element of the Galois field  $GF(2^J)$ , and  $g_j$ 's are the coefficients of  $g(x)$  with  $g_{2E} = 1$ . The generator polynomial defined above does not have symmetrical coefficients, i.e.,

$$g_j \neq g_{2E-j} \text{ for } j = 0, 1, 2, \dots, 2E.$$

A block diagram of an RS encoder which generates the remainder of  $f(x)/g(x)$  is given in Fig. 1. The switches in Fig. 1 are normally in the "ON" position until the last information symbol gets into the encoder. At this moment all switches are switched to the "OFF" position and the encoder is behaving like a long shift register. The output of the encoder is then taken from the output of the last shift register. Note that in Fig. 1  $2E$  multipliers are needed in the encoder.

To reduce the number of multipliers needed, a special class of the generator polynomial which has symmetrical coefficients was proposed by Berlekamp [10]. This generator polynomial is defined as

$$g(x) = \prod_{i=2^{J-1}-E}^{2^{J-1}+E-1} (x - \alpha^i) = \sum_{j=0}^{2E} g_j x^j$$

where

$$g_j = g_{2E-j} \text{ and } g_0 = g_{2E} = 1.$$

Note that since  $g_0 = 1$ , only  $E$  multipliers are needed. Thus using this new generator polynomial will reduce the number of multipliers required by one-half.

There are several schemes for interleaving the RS codes [5]. One scheme illustrated in Fig. 3 as "Interleave B" requires memory only for the parity-check symbols in the encoder is described as follows. In this scheme the input bits are grouped into  $J$ -bit symbols and transmitted in their natural order. However every  $I^{\text{th}}$  symbol belongs to the same codeword, where  $I$  is

the interleaving depth used. Thus  $I$  codewords make up such an interleaved code block. After the information symbols are transmitted, the parity-check symbols of each interleaved codeword are then transmitted.

If interleaving is used, then the encoder logic structure is the same as shown in Fig. 1, except now each  $J$ -bit shift register is replaced by an  $I \times J$  - bit shift register. As an example, a block diagram of a (255,223) RS encoder with interleaving level  $I$  and generator polynomial

$$g(x) = \prod_{i=1}^{143} (x - \alpha^i)$$

where  $\alpha = 2$  in  $GF(2^8)$ , which is generated by the primitive polynomial

$$x^8 + x^4 + x^3 + x^2 + 1,$$

is shown in Figure 4. Note that a generator polynomial with symmetrical coefficients is used here to save multipliers.

### III. Symbol-Slice VLSI RS Encoder Architecture

A finite field multiplication is a quite complicated operation. There are basically three techniques for implementing a finite field multiplication. The first technique is to use log and antilog tables stored in read-only memories (ROM's) [4]. The second technique is to use a linear feedback shift register type of approach [9]. The third technique is to use the property of the trace in a finite field to form a smaller ROM look-up table [10]. Due to the advent of LSI ROM technology, techniques 1 and 3 are usually used in an RS encoder design optimized for discrete IC's. As an example a 400 KHZ (255,233) RS encoder using the Berlekamp's approach [10] requires only around 40 CMOS IC's.

When one is interested in further drastic reduction of the power and size of an RS encoder for high speed applications, one has to consider VLSI implementations. An RS encoder design optimized for discrete IC's usually does not have a modular structure. Hence when one uses such a architecture for VLSI layout, one has the following problems:

- (1) The design is too big to be put on one chip,

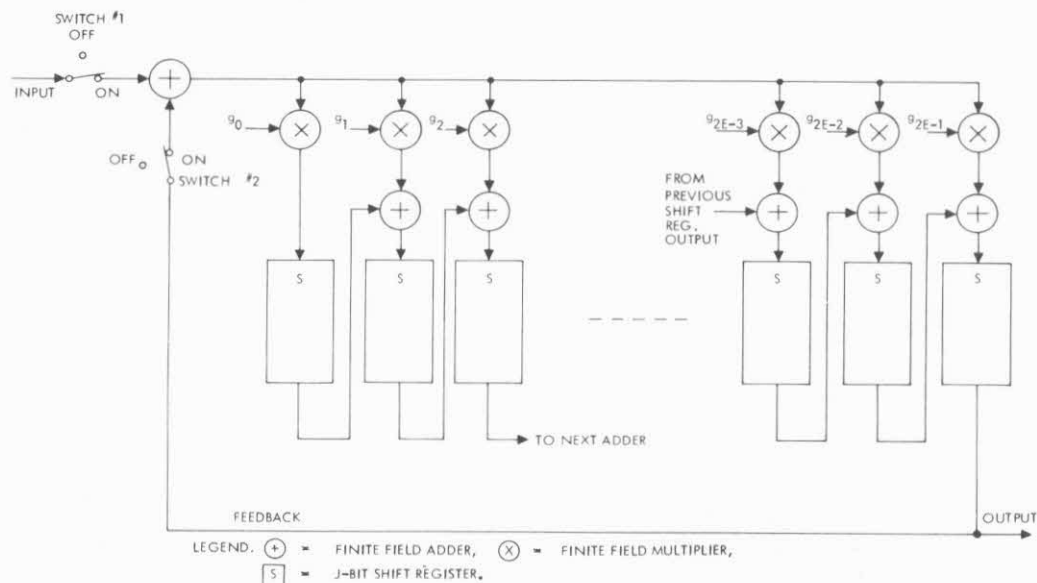


Figure 1. A Block Diagram of a  $(2^J-1, 2^J-1-2E)$  RS Encoder Using a Conventional Generator Polynomial.

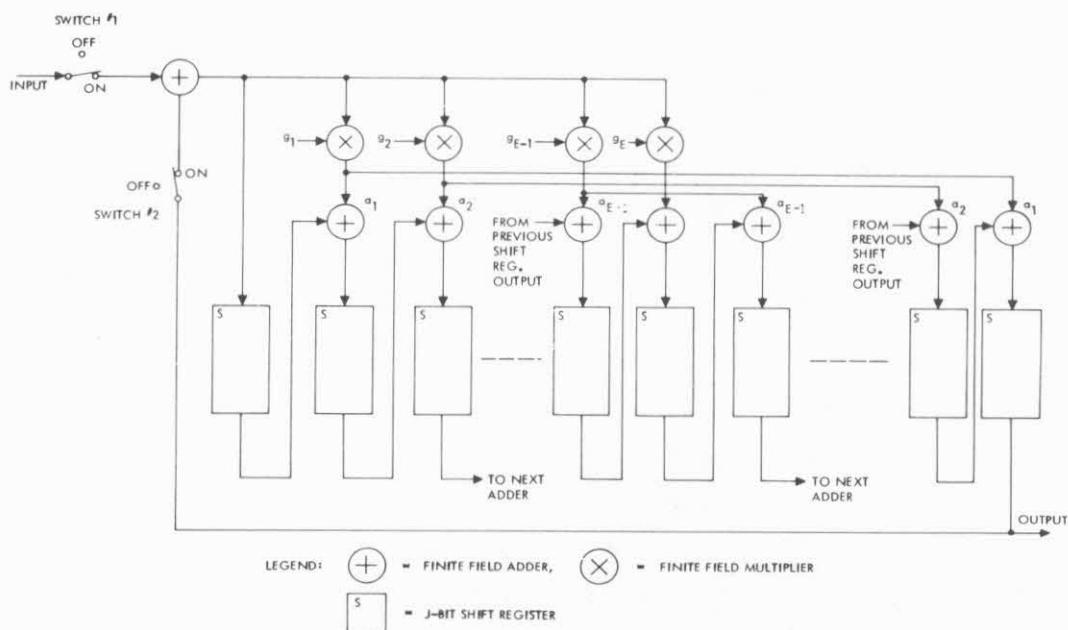


Figure 2. A Block Diagram of a  $(2^J-1, 2^J-1-2E)$  RS Encoder Using a Generator Polynomial With Symmetrical Coefficients.

- (2) If a multiple-chip approach is used, then one needs several chip designs, where each chip has an impractical number of input/output pins.
- (3) The design is not modular. Therefore the design is not easy to adapt to other RS code parameters.

Hence there is a need to find a VLSI logic structure which can alleviate the above problems.

The repetitive architecture of the RS encoders shown in Figs. 1, 2, and 4, suggested that a symbol-slice type of VLSI chips, each one consisting of a fixed portion of the encoder, may be cascaded to form a complete RS encoder. Also to reduce the VLSI chip size, RS encoders using generator polynomials with symmetrical coefficients are preferred. Hence we will put emphasis on this type of VLSI RS encoder.

As an example, we will design a VLSI encoder chip for a 255-symbol, 8-bit per symbol, 16-error-correcting, RS code with an interleaving level of 5. The primitive polynomial used is

$$x^8 + x^4 + x^3 + x^2 + 1$$

The generator polynomial for each codeword is

$$g(x) = \prod_{i=112}^{143} (x - \alpha^i)$$

where  $\alpha = 2$ . The encoder logic structure for the above RS code parameter is identical to the one shown in Fig. 4, except now  $I = 5$ . There are several ways of partitioning the RS encoder into four sections. One way which requires a minimum of input/output pins is to include four rows of logic shown in Fig. 4 into one section. Each section is then realized by a universal VLSI RS encoder chip. The logic structure of this universal chip is shown in Fig. 5. The entire VLSI encoder system, which consists of four identical VLSI RS encoder chips cascaded and properly interconnected together is shown in Fig. 6. Each VLSI RS encoder chip has 24 pins. A detailed description of the VLSI RS encoder chip and the entire VLSI RS encoder system is described as follows:

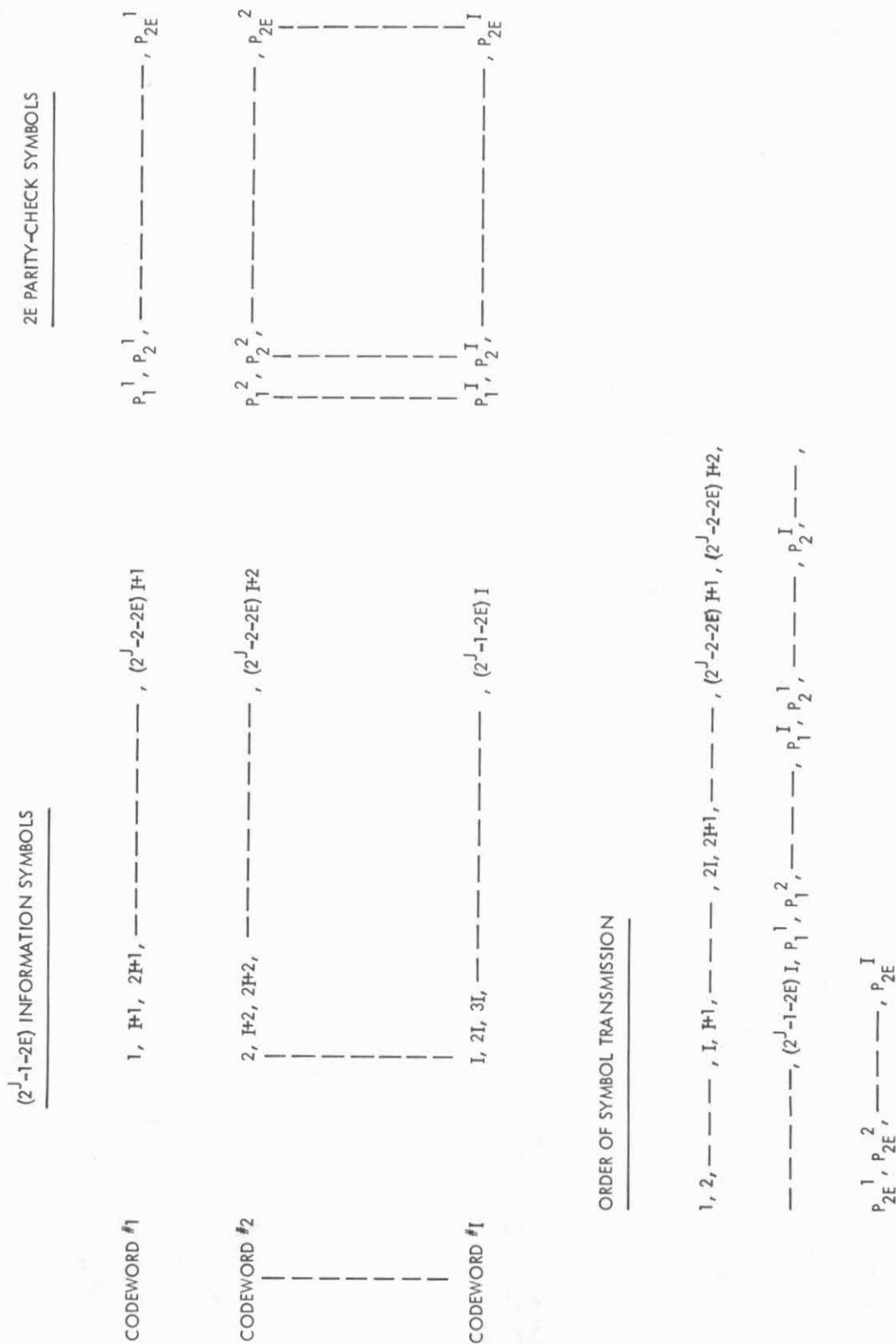


Figure 3. Code Array Structure and Order of Symbol Transmission For Type B Interleaving, Where Interleaving, Where Interleaving Level = I

### 3.1 Generator Polynomial Coefficients Table

Since a generator polynomial  $g(x)$  with symmetrical coefficients is used, the coefficients of  $x^0$  is always 1. Hence there is no need for a multiplier to operate on this coefficient (see Fig. 4). Consequently if the new generator polynomial is used, then one only needs  $E/N$  multipliers on each VLSI chip, where  $E$  is the error correcting capability of the code and  $N$  is the total number of chips required in a VLSI encoder system. In the design example,  $E = 16$  and  $N = 4$ . Hence 4 multipliers are used on each VLSI RS encoder chip. To make the VLSI chip universal, all distinct coefficients except 1 of the generator polynomial are stored in a read-only memory on the chip. In general, an  $E \times J$ -bit table is needed. In the design example  $E = 16$ ,  $J = 8$ . Hence a  $16 \times 8$ -bit table is selected. The outputs of an  $E \times J$ -bit table is fed into  $N$ ,  $E/N$ -to-one multiplexors. The outputs of the multiplexors are selected by  $\log_2 N$  input pins called the "chip select" or "G select" pins. These outputs are then fed into the inputs of the  $E/N$  multipliers. In the design example two "G select" pins (pins 22 and 23) and four, four-to-one multiplexors are used. The  $16 \times 8$  table and the multiplexors can easily be implemented by four,  $4 \times 8$  ROM, with G select signals as the address control lines of each ROM. The coefficients of  $g(x)$  selected on each chip are shown in Fig. 6.

### 3.2 Finite Field Multiplier

Next we will discuss the architecture of the finite field multiplier. To connect the multiplier properly between chips and at the same time minimize the number of input/output pins used, a linear feedback shift register type of multiplier [9] rather than a ROM table lookup type of multiplier [4] is adopted. The multiplier used is of a serial-parallel type. The  $J$ -bit generator polynomial coefficient is read out from the  $E \times J$ -bit ROM table and fed into the multiplier  $J$ -bit in parallel whereas the other input, generated by the feedback input (pin 2) "ANDed" with the feedback enable (pin 1), is fed into the multiplier bit-by-bit in serial.

The output of the multiplier is loaded into an 8-bit shift register in parallel at the end of every 8th bit clock (1 symbol clock time). The parallel data is serialized by this shift register. The most significant bit (MSB) output of this shift register is added with the MSB of the 40-bit shift



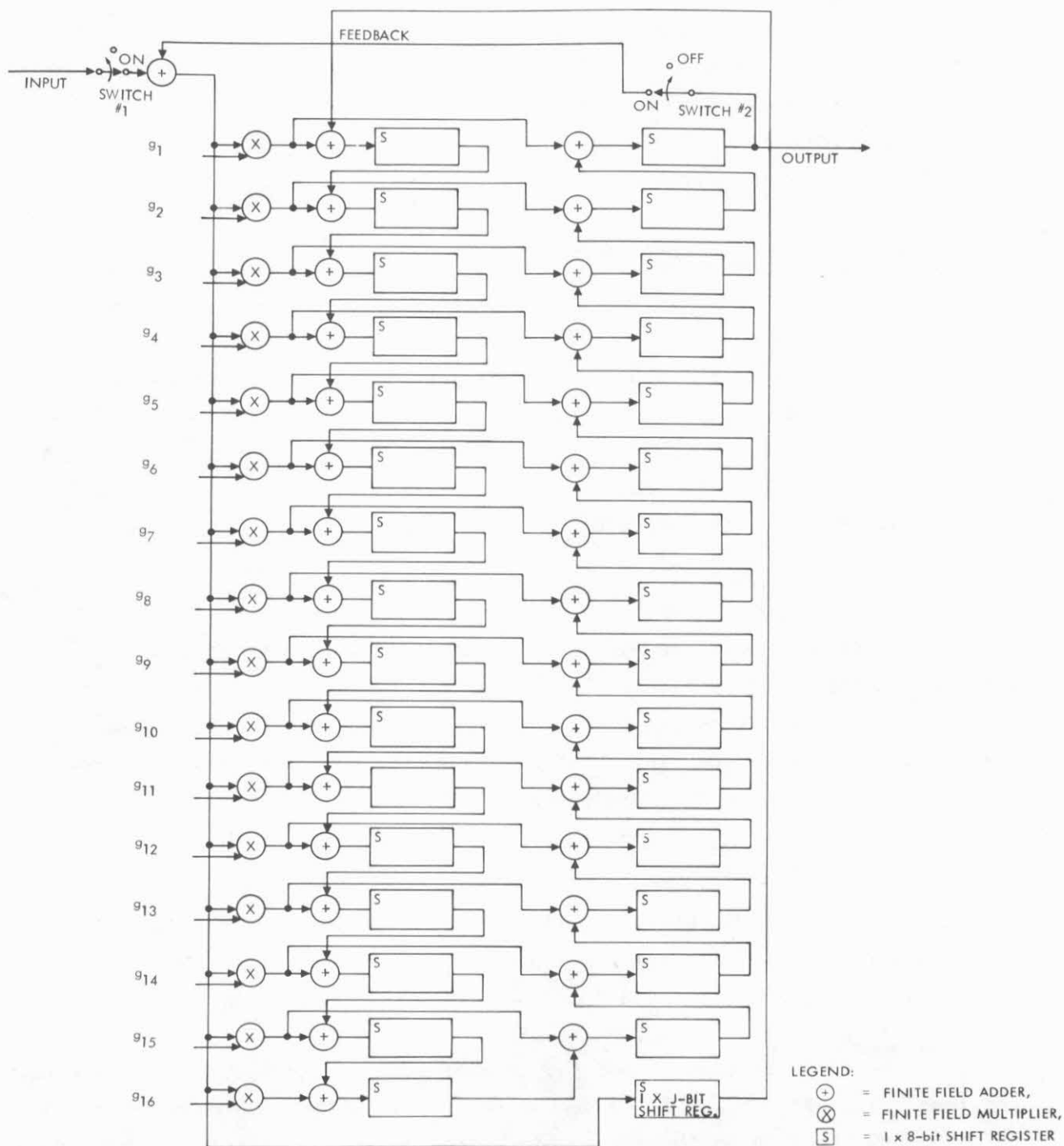


Figure 4. A Block Diagram of a (255,223) RS Encoder with Interleaving Level I and New Generator Polynomial.

register, which is either on the same chip or on a different chip, and the resulting data is shifted into the least significant bit (LSB) of the next 40-bit shift register. The adder is implemented by a two-input EXCLUSIVE-OR gate (there are eight EXCLUSIVE-OR gates on each chip). Each adder takes an input from a multiplier output which is either on the same chip or on a different chip, depending on the coefficients of the generator polynomial.

### 3.3 Input and Feedback Control Switches

The switch #1 shown in Fig. 4 is implemented by an AND gate on the chip with the bit-serial data input (pin 3) and the feedback enable signal as the two inputs. The feedback enable signal is provided by an external modulo 255 counter which is driven by a clock equal to the bit clock divided by 40 (see Fig. 6). This signal is true when the counter is counting from 1 to 223; otherwise it is false. The output of switch #1 is added with the MSB of the 40-bit shift register S5 output on the same chip to generate the feedback output signal (pin 5). This signal is redundant in all but the first chip (see Fig. 6).

The switch #2 shown in Fig. 4 is implemented by an AND gate on the chip with the feedback enable and feedback input signals as the two inputs. The feedback input signals on all chips (pin 2) are connected to the feedback output signal (pin 5) on the first VLSI chip. The output of switch #2 is fed into all multipliers on the chip bit-by-bit in serial.

### 3.4 Input/Output Data Connections

There are eight input/output lines on each chip. Of these eight lines, four lines (pins 8, 9, 11, 17) are input lines and the remaining are output lines (pins 6, 7, 10, 13). Pin 8 is normally connected to pin 6 on the same chip except for the last chip, where pin 8 is grounded. Pins 7 and 9 are normally connected to pins 17 and 13, respectively on the next chip except for the last chip, where pin 7 is connected to pin 11 on the same chip and pin 9 is connected to pin 4 on the first chip. Thus one has a railroad type of data connections between chips. The reason for connecting pin 4 on the first chip to pin 9 on the last chip is a consequence of an inherent 8-bit multiplier delay. To replace the multiplier on the  $x^0$  position by the switch #1 output,

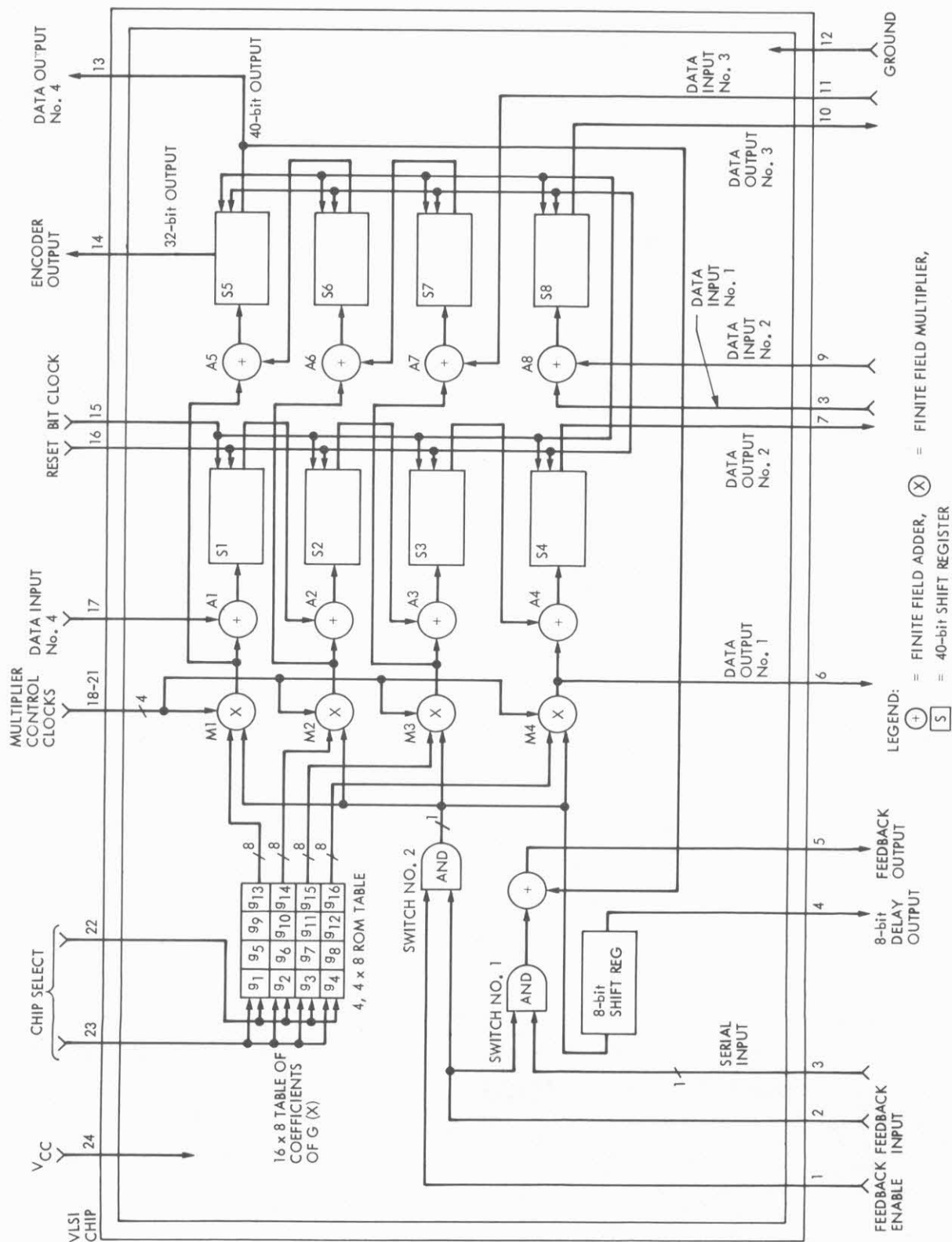


Figure 5. VLSI RS Encoder Chip Logic Structure

one needs to delay this output also by 8 bits to lineup the bits. For other chips besides the first chip, the 8-bit register outputs are not used.

The encoder output is taken 8 bits earlier from the MSB of the last 40-bit shift register on the first chip. This is because when the last bit of the last symbol in the information part of the code is shifted into the encoder, the contents of each multiplier are loaded into the 8-bit output shift registers waiting to be added with the MSB of the 40-bit shift registers. These 8-bit symbols in the multiplier output registers are actually belong to the fifth codeword in the interleaved code array. However the parity-check of the first codeword is already being computed and now sitting 8 bits from the MSB's of each 40-bit shift registers. Hence the 32-bit output (pin 14) of the last 40-bit shift register on the last chip is the output of the VLSI RS encoder system. This output is taken when the external modulo 255 counter is counting from 224 to 255. Since at these times switches #1 and #2 are turned off, the entire VLSI encoder system is behaving like an  $1 \times 2 \times J$ -bit (e.g.  $5 \times 32 \times 8$ -bit in the design example) shift register. Thus the  $5 \times 32$ , 8-bit parity-check symbols are read out from the VLSI RS encoder bit-by-bit in serial and appended to the  $5 \times 223$ , 8-bit information symbols.

Another way to partition the RS encoder shown in Fig. 4 into four sections is to include 8 rows of logic in each column into one section. Each section is then realized by a universal VLSI RS encoder chip. The logic structure of this chip is very similar to the one shown in Fig. 5 except now one has to provide output pins to all multipliers and input pins to all adders on the chip. Hence this chip uses 6 more input/output pins than the one shown in Fig. 5. The interchip connections between adders and multipliers are similar to the pyramid type of connections shown in Fig. 2.

#### IV. VLSI RS Encoder Chip Size Assessment

It is estimated that it is impossible to put the entire VLSI RS encoder chip design on a  $235 \times 235$  mils CMOS/bulk VLSI chip using a  $7\mu$  standard cell approach on all logic. However, if one uses custom RAM and ROM cells design to implement the 40-bit static shift registers and the  $16 \times 8$  table and  $7\mu$  standard cell design for the rest of the logic, then it is possible to have a one-chip design.

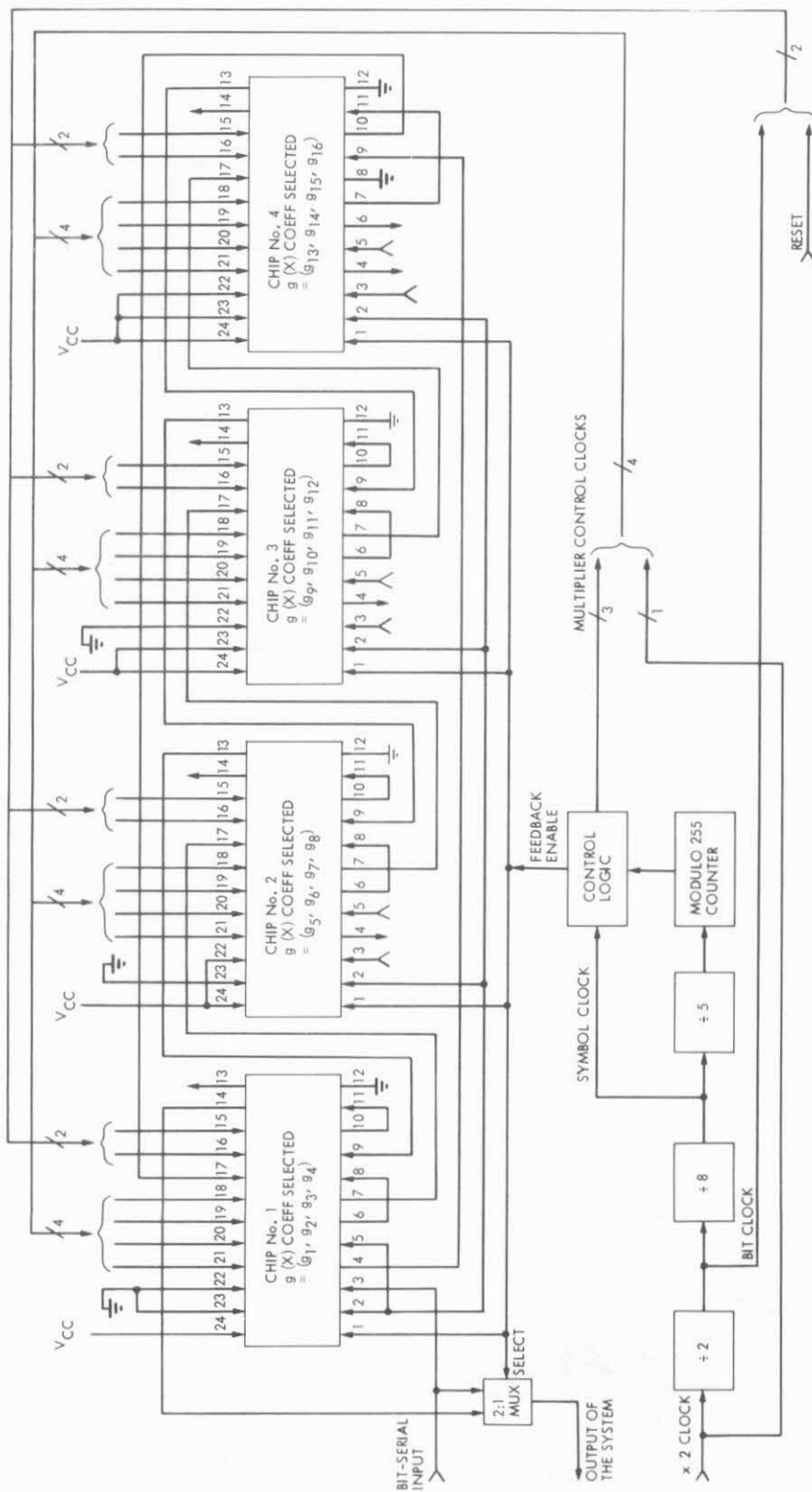


Figure 6. VLSI RS Encoder System Diagram

## V. Performance of the VLSI RS Encoder System

For design verification, both the shift register version and the RAM version of the VLSI RS encoder chip and VLSI RS encoder system are implemented using discrete CMOS IC's and are now operational. The throughputs of these two versions are 800K bits/sec for the shift register version and 200K bits/sec for the RAM version. These throughputs are expected to go much higher if the actual VLSI encoder chips are used.

## VI. Conclusions

We have just shown the logic structure of a symbolic-slice VLSI RS encoder chip and the VLSI RS encoder system built by these chips. A design example has been given for a (255,223) VLSI RS encoder chip and VLSI RS encoder system. It has been shown that an RS encoder consists of four identical CMOS VLSI RS encoder chips connected together may replace around 40 CMOS IC's required by an encoder design optimized for discrete IC's. Besides the size advantage, the VLSI RS encoder also has the potential advantages of requiring less power and having a higher reliability.

References

- [1] I.S. Reed and G. Solomon "Polynomial Codes Over Certain Finite Fields," J. Soc. Indust. Appl. Math., 8, pp. 300-304.
- [2] G.D. Forney, Concatenated Codes, the MIT Press, Cambridge, Mass., 1966.
- [3] J.P. Odenwalder, "Optimum Decoding of Convolutional Codes," Ph.D. dissertation, Syst. Sci. Dept., Univ. of Calif., Los Angeles, 1970.
- [4] J.P. Odenwalder, et al. "Hybrid Coding System Study," submitted to NASA Ames Research Center by Linkabit Co., San Diego, Calif., Final Report, Contract No. NAS-2-6722, Sept. 1972.
- [5] R.F. Rice, "Channel Coding and Data Compression System Considerations for Efficient Communication of Planetary Imaging Data," Technical Memorandum 33-695, Jet Propulsion Laboratory, Pasadena, CA., June 1974.
- [6] R.F. Rice, "Potential End-to-End Imaging Information Rate Advantages of Various Alternative Communication Systems," JPL Publication 78-52, June 15, 1978.
- [7] A. Hauptschein, "Practical, High Performance Concatenated Coded Spread Spectrum Channel for JTIDS," NTC '77, pp. 35:4-1 to 4-8.
- [8] K.Y. Liu and K.T. Woo, "The Effects of Receiver Tracking Phase Error on the Performance of the Concatenated Reed-Solomon Viterbi Channel Coding System," NTC '80, pp. 51.5.1 to 51.5.5.
- [9] W.W. Peterson and E.J. Weldon, Jr., Error Correcting Codes, The MIT Press, 1972.
- [10] E.R. Berlekamp, "Better Reed-Solomon Encoders," presented at Calif. Inst. of Tech. EE Seminar, Pasadena, CA., Dec. 12, 1979.





# Communications for Next Generation single chip computers<sup>\*</sup>

David R. Smith<sup>#</sup> and Douglas Chan

State University of New York at Stony Brook

## Abstract

It is the thesis of this report that much of what is presently thought to require specialized VLSI functions might instead be achieved by combinations of fast general purpose single chip computers with upgraded communication facilities. To this end, the characteristics of applications of this nature are first surveyed briefly and some working principles established. In the light of these, three different chip philosophies are explored in some detail. This study shows that some upgrading of typical single chip I/O will definitely be necessary, but that this upgrading does not have to be complex and that true multiprocessor-multibus operation could be achieved without excessive cost.

## I. Example applications

Without doubt two of the most important computer applications of the coming decade will be graphics and speech processing. In turn, two principal characteristics of these applications are that the computation is often divisible into modular components and that they are often real time driven. Hence the requirement for speed which also points in the direction of specialized hardware. Examples may be found in the recent literature:

1. In a recent graphics processor [3], 12 copies of a specialized chip in combination are proposed to process graphic images for rotation, skew, translation, and various kinds of scaling and clipping. The computation rates of the component modules do not seem incompatible of the performance expected of next generation single chip computers.

2. A recent text-to-speech system [4], employs two general purpose processors feeding into specialized hardware. At the anticipated three character per second input rate, the second processor communicates data in packages of between 2 and 130 bytes every 10 milliseconds.

---

<sup>\*</sup>Supported by General Instrument Corporation.

<sup>#</sup>Visiting E.E. Dept., Stanford University, Spring 1981.

3. As can be seen from a recent survey, word recognition and continuous speech recognition systems [5], while still largely in the laboratory stage, are clearly developing along modular lines. The acoustic analysis module of one system [6] (itself done by FFT and divisible into modules), passes samples of 14 functions every 10 milliseconds on to the phonetic analysis module. Another [7], using software modules on a last generation mainframe, quotes ratios of cpu time to speech time of the following modules in series: Signal analysis 3:1, Spectral similarity 2:1, Region definition 2:1, Boundary placement 1:1.

It is to be noted that the information flow in all the above systems is unidirectional. As a contrast to this, a very common structure with the present generation single chips is to employ them as the slaves in a master-slaves configuration, eg. inside an intelligent terminal. In this case the communications are sometimes bidirectional. Another recent example of the master-slaves bidirectional configuration [8] uses special purpose processors to scan superimposed codes of a data base index.

All these applications are of course an illustration of the fact that parallel processing applications are emerging first in the fixed purpose realm. Although the ones mentioned here do not constitute a large number, they perhaps serve to give some idea of the features desirable in single chip computers which would serve them.

## II. Working principles

In this section we set down what seem to be reasonable principles to be drawn at the outset concerning inter-chip communications in this context and some of the reasons for preferring them.

1. The processors will manipulate data in 16 bit words. In single chip computers the 16 bit word length will play for the next several years, and is sufficient for digital representation of analog quantities.

2. The number of pins per package is strictly limited and will not greatly differ from current practice.

3. Communications will be assumed to be word parallel. This is somewhat a choice of convenience if other considerations permit. Parallel ports are required anyhow for uniprocessor applications, and while serial communication could still be an added alternative, a decision to employ parallel communication will avoid the necessity for repetitive conversions through on-chip UART devices. In addition the use of the parallel ports would avoid further disparity in the estimated two orders of magnitude which separates the speed of communications internal and external to the chip [9].

4. There is no need of an address bus. Present designs for multibus architectures (eg. Multibus, Versabus, etc) are based on the arbitration of complete parallel bus structures, including the address bits [eg. 10].

In our context however this does not seem to be necessary. This is because we expect each communication (of parameters or results) between components of a modular computer structure to consist typically of a message or packet incorporating a number of words. This is shown in the applications described above. It has also been our experience with simulated applications for the Stony Brook Multicomputer, and in fact incorporated into the design of the kernel [11]. So if a whole package of words is going to a single destination, then an address bus would lie idle for all data words after the first. A destination address may as well be incorporated in the package header.

5. Bus connectivity: The next question to be addressed is how many distinct communication ports should emerge from a computer node. Clearly one is sufficient (eg. Ethernet), but in the present context not very interesting since the consequent restriction to a single time shared bus limits the concurrency possible. Performance is one of our concerns. If a condition were made that the processor internal bus must be available at the pins for testing purposes, then a minimum of 2 ports per node would be necessary, since distinct processor busses could not be directly interconnected. We will assume here that this is not the case, - that internal chip logic can be arranged so that the processor internal bus is available for testing at chip reset time, but after that a mode change can be effected which transforms that port into a buffered communication facility.

What then should be the connectivity of a node? Hardware for the X-tree project [12] is being constructed with 5 ports per node, although this would impact the pin resources of standard single chip packages. Also it is not clear that such a port multiplicity is necessary to be able to construct the network topologies considered in the X-tree literature, or to accommodate the bus loading of typical applications. This is because in fact, communication ports can be configured with appropriate control signals so as to be shared, and the apparent connectivities of modular computer graphs seen in the literature does not have to correspond to the number of bus ports coming off the chip. For example, the Stony Brook Multicomputer [11], though normally drawn as a graph having up to 6 edges to a node (fig. 1a), would, if implemented with single chip computers, be most conveniently partitioned so as to use only 3 port types per node (fig 1b). Simulation studies of typical applications [13] did not show bus loading as one of the limiting performance factors of the network.

Let us therefore consider the number of busses emerging per node. If this is two, then the maximum communication concurrency of the whole network is equal to the number of computing nodes ( $n$ ), and occurs when they are all strung out in a simple pipeline as in figure 2. This assumes the use of buffered or DMA type ports able to operate independently of the CPU, otherwise, with program controlled ports, the maximum would be  $n/2$ . Of course it would also be less if some busses were incident on more than two nodes. Suppose now that the number of busses incident on a node is three. Then as far as the bus hardware is concerned the communication concurrency of the network might exceed the number of computing nodes. But

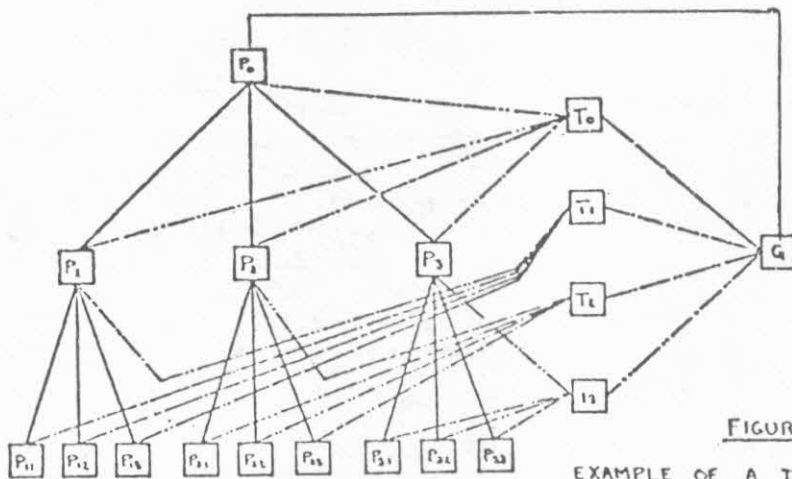


FIGURE 1(a)

EXAMPLE OF A DOUBLE TREE NETWORK (SUSB)

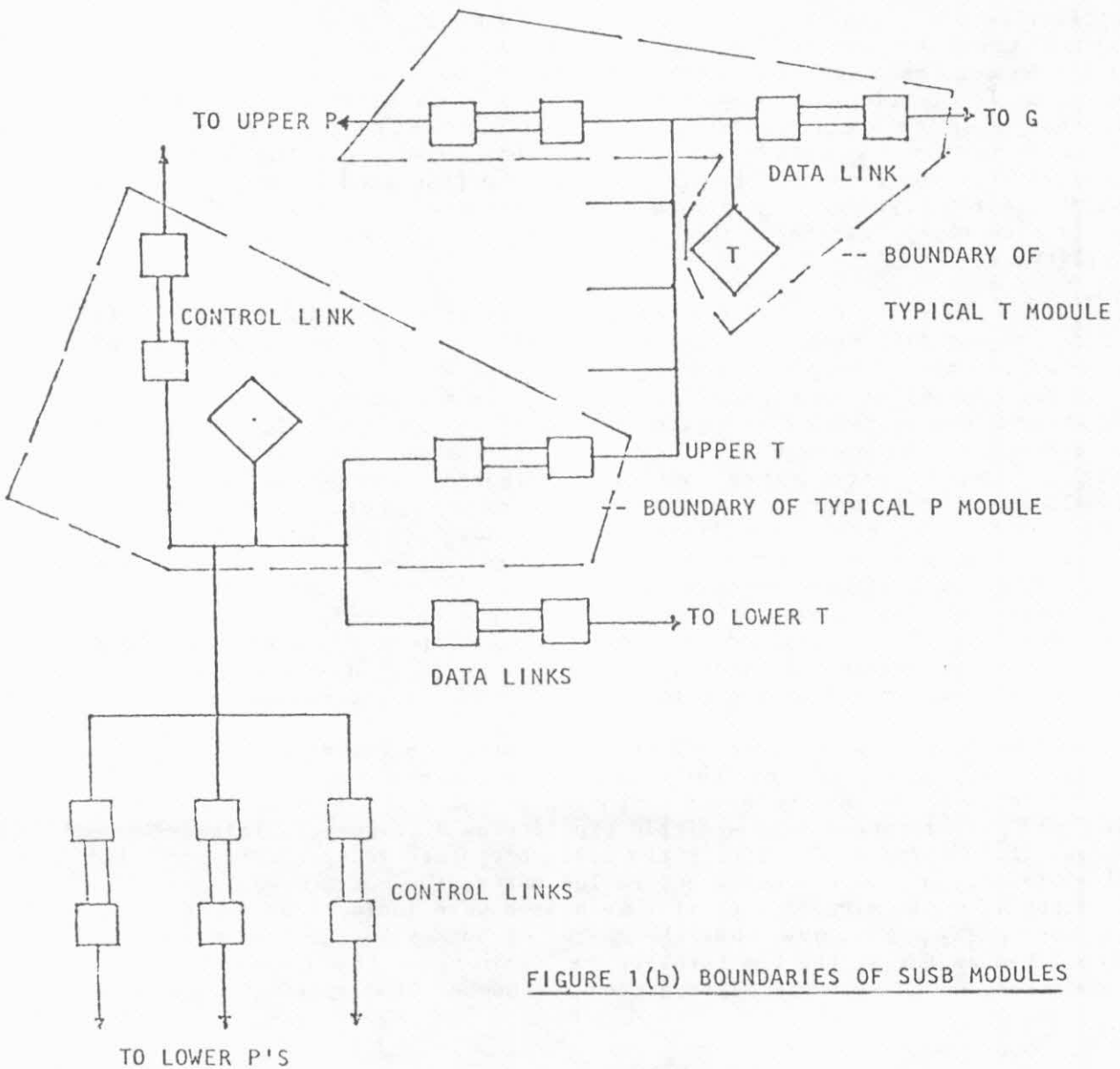


FIGURE 1(b) BOUNDARIES OF SUSB MODULES

unless the node computation is something rather trivial, it seems doubtful that results would be produced at a sufficiently rapid rate to justify anything like such a high communication/computation ratio. This supports the argument, already almost dictated by current pin limitations, that the number of bus ports should not exceed two. In paragraph 7 below, directionality arguments lead us to the conclusion that the number of ports should be exactly two.

6. Speed-up by pipelining and synchro-parallelism: For the internal structure of processors it is well known that there are two important recourses for when it is desired to achieve a computation rate which exceeds the speed of the hardware available. These have been termed 'pipelining' and 'synchro-parallelism' [14]. They are illustrated in the context of  $n$  computer chip nodes in figures 2 and 3.

In the first, the serial combination, each computing node performs a part of the total computation and passes the intermediate results on to the next stage. In the simplest arrangement, the constituent computations all take the same amount of time,  $T$ , to complete. Then although a single computation still takes time  $nT$ , the results of repeated computations are streamed out at time intervals of  $T$ . If the constituent computations take differing amounts of time, then the technique may still be effective, but handshaking controls must be provided to make the faster components wait, and the overall results will be produced at intervals  $T_{\max}$  corresponding to the slowest.

In the second, (fig 3), a parallel combination utilizes a staggered computation with input parameters and output results sequenced on shared busses. In the simplest form the constituent modules could be doing identical computations, or similar computations with different internal coefficients, and would therefore all take the same amount of time  $T$ . Again the effective computation rate is speeded up  $n$  times, in this case one result being produced every  $T/n$  secs. In a more complex case, the computations might take differing amounts of time, and then again the computation must be controlled to make the faster ones wait. In a more complex case still, serial networks of parallel stages might be indicated to more closely match the speeds of the pipeline stages, or alternatively, parallel networks of serial combinations, etc.

The important point is that the communication facilities that we propose should accommodate these likely to be encountered cases.

7. Bidirectional vs unidirectional ports: From the fixed applications literature we conclude that unidirectional communications are those most called for, although bidirectional communications are an important minority which must be accommodated. However, the bidirectional facility would be complex for the message based type busses we have outlined so far. It would require some type of semaphore system to control the

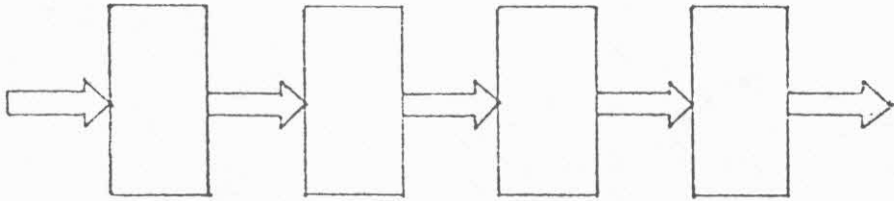


FIGURE 2 PIPELINE OR SERIAL CONNECTION

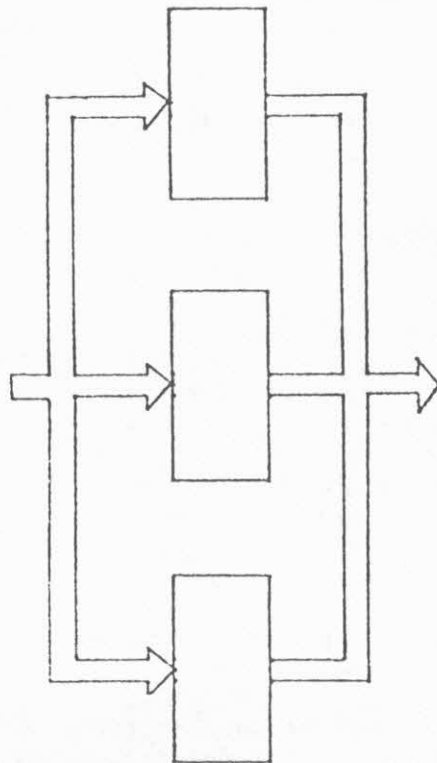


FIGURE 3 SYNCHRO-PARALLEL CONNECTION

intended bus master and direction of data transfer and to resolve collisions. Once established, communications on a fixed route and direction might proceed efficiently, but the protocol to change these would be a burden in terms of pins and time. The resulting overhead would then lie unused on what seems to be the majority of unidirectional communication applications. The alternative would be to employ a unidirectional transfer protocol on each port which was as simple in design as possible, as long as it could be shown that bidirectional applications could still be serviced (at the cost perhaps of using one or two extra chips). This will be the approach adopted here.

### III Design for upgraded single chip computer,- version 1

A block diagram of a proposed design for an upgraded single chip computer is shown in figure 4 and will be used as a first model.

The large shaded block in the interior labelled SCC corresponds in function to a single chip computer of existing design. This will have the advantage that the design and debugging of this part has already been done and hopefully, partially amortized. As shown in our diagram we assume only that this part has two bidirectional I/O ports labelled A and B, and two interrupt inputs T0 and T1.

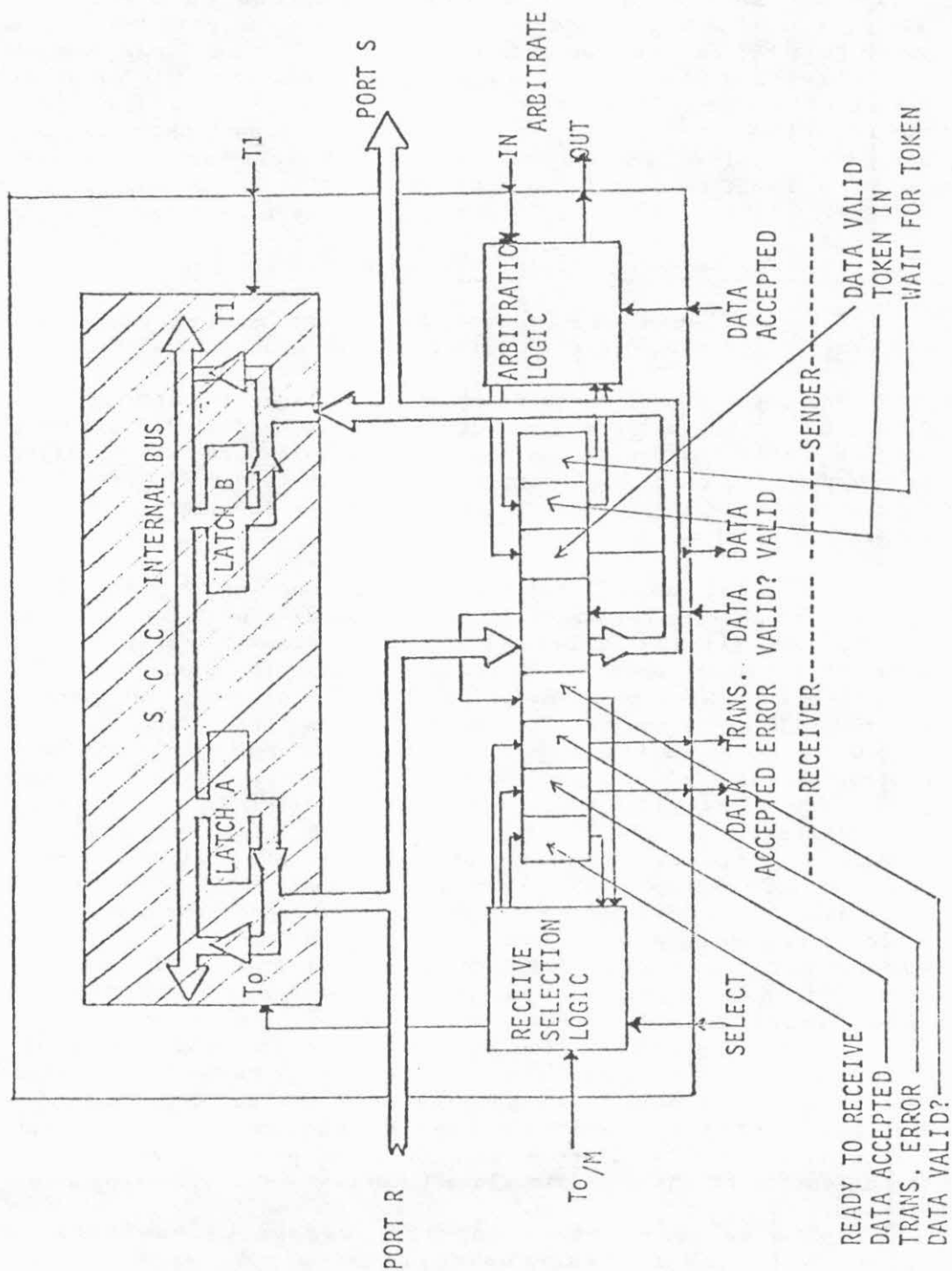
The additional logic circuit functions which give the new chip its upgraded capability are shown in the surrounding unshaded area of figure 4. Besides all the original pin connections to the SCC (which are led straight through), there are 8 additional pins for the added functions. This would increase the typical current 40 pin package to 48 pins which is not regarded as excessive. The added logic has three sections. Going from right to left in the figure, these are the arbitration control, control-status register, and the receive selection logic. The internal I/O ports are connected through to the external I/O ports S and R (for send and receive). Since the internal ports are bidirectional, while the external ports are unidirectional, we can utilize the remaining two functions to read and write the new control-status register with the existing SCC instruction set as shown. The need for the new arbitration logic arises because we are here absorbing the functions of the separate bus arbitration chip of existing commercial designs. It will operate on a simple daisy chain principle and be used chiefly for arbitration of the send functions. The receive selection logic is responsible for enabling the communication pathway leading to this chip from the send port of another. When the module is not so selected, its receiver port and handshake lines should be in the high impedance state. When the READY FOR RECEIVING flag is set, and the SELECT input is active, the receiver selection logic will interrupt the SCC using pin T<sub>0</sub>, which will invoke the attention of the communications kernel.

In two matters concerning this design we have been a little conservative, but these could presumably be adjusted later as the technology permits. Current instruction sets typically include a repeat function, which as applied to I/O, could be capable of streaming words at



## UPGRADED SINGLE CHIP MICROCOMPUTER

FIGURE 4





an exceedingly rapid rate. Although we envisage multiple computer chips in close proximity, say on the same board, we are unsure that this rapid rate could be correctly synchronized over a variety of parallel and serial connections, and have therefore opted for the usual double handshake by word (see the data valid and accepted pins in figure 4). Secondly, it would be entirely feasible even now to insert FIFO buffers between our internal and external ports in each direction. This might be particularly effective in combination with the new 'repeat I/O' instructions, and if the message sizes were standardized. We will examine a configuration similar to this in a later section.

#### Communication between modules

Figure 5 shows the format of the message block. The first word contains an eight bit field which carries addressing information about the receivers, followed by eight redundant check bits. This is similar to a scheme suggested by R.B. Kiebert for the Stony Brook Multicomputer. It is to provide message synchronization in the absence of input and output control pins allocated to this purpose. The kernel of the receiver will output on its ERROR pin in the event of communication error or if it gets out of message synchronism with its sender. On receiving this through its  $T_{O/M}$  interrupt input, the kernel of the sender will restart with a message header. The second word contains source address and message length fields. The whole message can also be protected with a longitudinal check with errors reported on the same pin if desired.

The most straightforward communication method supported by the above modules is illustrated by the bus connection in figure 6.

All the modules which have their send ports connected to the bus will have their ARBITRATION IN and OUT pins strung in a simple cyclic chain. As shown an active pulse must be inserted into the chain at reset time to start things off. The arbitration logic in each module is responsible to check for the receipt of a pulse (token) at its input. If the WAIT FOR TOKEN flag is set by the kernel, then the TOKEN-IN bit will be set. Else the token will be passed on immediately by the hardware to the next module. This will make possible the simple sequencing of the synchro parallel connection of figure 3 (WAIT FOR TOKEN flags always set), or in a more asynchronous situation, a rapid round robin determination of which module is requesting service and is next in line.

On the receiving side, the select line input of each receiver is connected from a different line of the data bus and all the DATA ACCEPTED lines of receivers attached to this bus are connected together in WIRED-AND fashion. This shown in figure 7. When quiescent, this output is in the high impedance state. However, when a receiver is selected it changes this output to logic true or false, depending on its READY FOR RECEIVING flag. To the sender, its DATA ACCEPTED? input will not be true until all the receivers it has selected are ready. This makes possible the following communication modes to up to eight receivers; PUBLIC BROADCAST: all receivers in the subset designated by the sender receive the message;

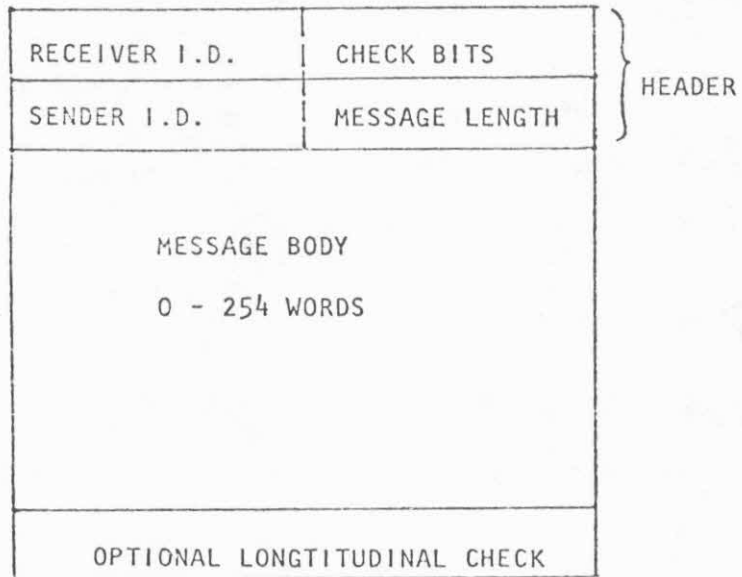


FIGURE 5 MESSAGE FORMAT

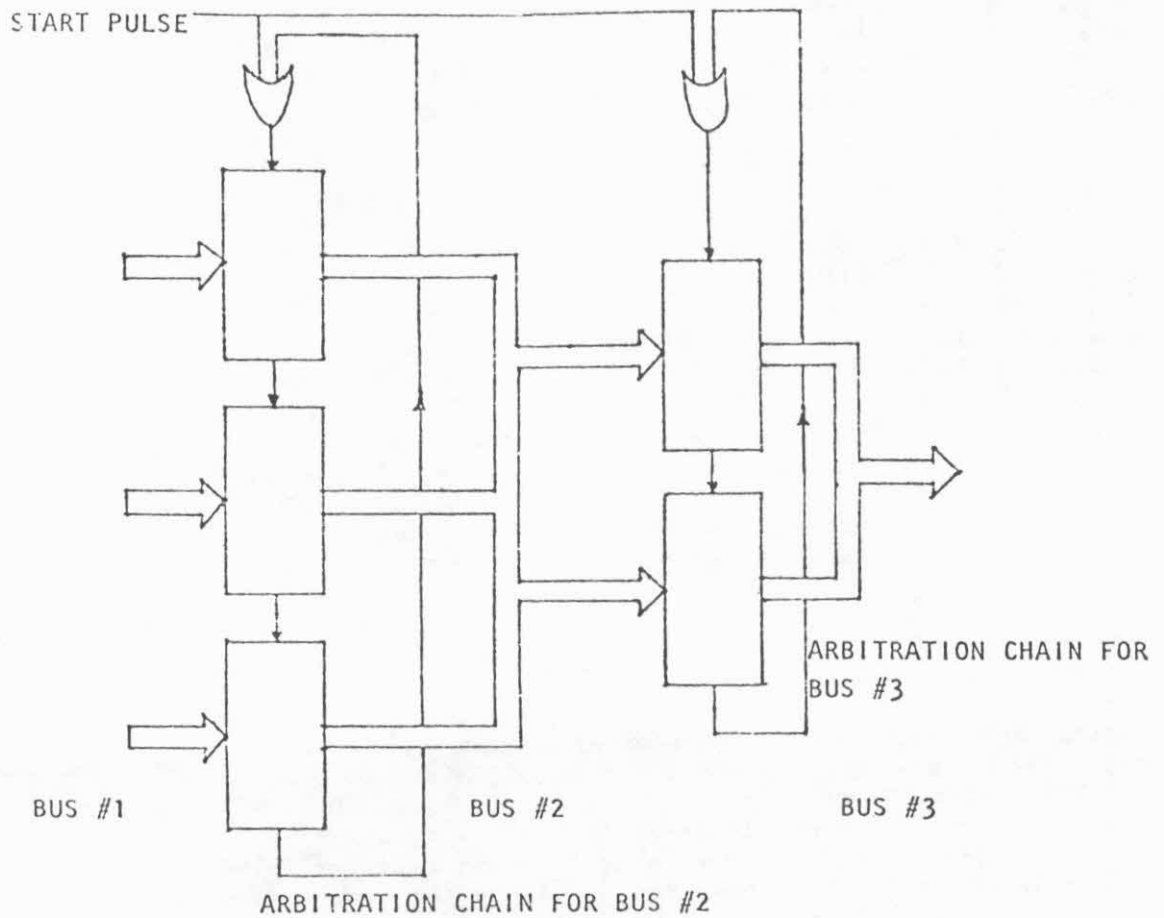


FIGURE 6 ARBITRATION CONNECTIONS

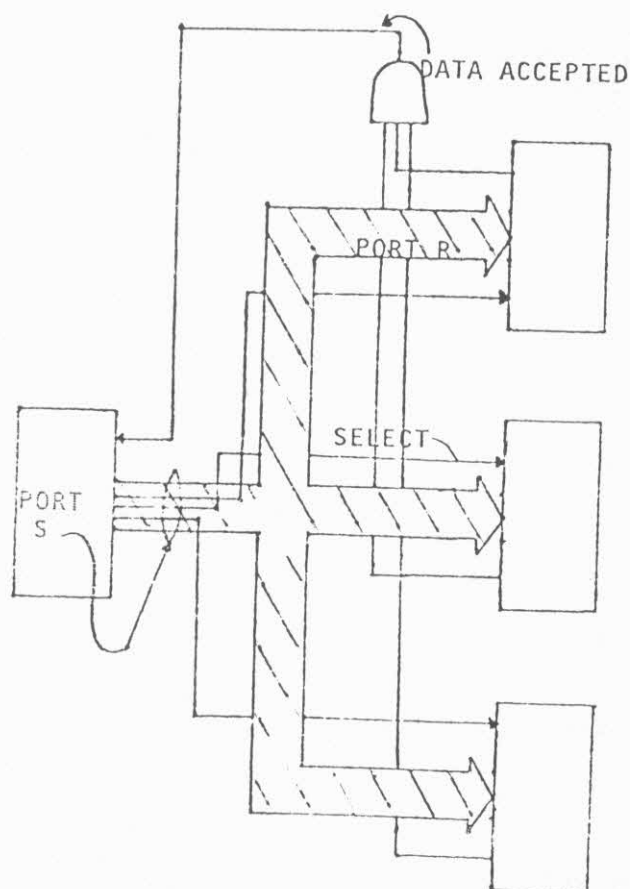


FIGURE 7  
SELECT AND HANDSHAKE  
CONNECTIONS

FIRST READY: the next receiver to be ready will receive the message. For both of these cases the receiver selection is mediated by the header word of the communication message block. In the FIRST READY case however only the rightmost bit in the header would be initially set and shifted one bit left after each timeout until either a ready receiver is found or all the receivers have been exhausted. In the latter case the kernel would then abort that transmission attempt and go to sleep until the next cycle of the daisy chain token. Thus FIRST READY is just a multiple application of the PUBLIC BROADCAST with a subset size of one.

In applications in which larger fan-outs are called for, this may be obtained by simply connecting the chips in the form of a tree. In this way one extra layer of 8 chips could achieve a fanout of 256 and so on. This is the same technique commonly used in multiplexer and decoder trees.

A detailed listing of the send and receive protocol sequence is given in appendix 1.

#### Multi-computer configurations.

Clearly the design we have outlined will be most efficient when used in simple serial or parallel unidirectional structures as in figures 2, 3, and 6. However, if bidirectional communication is desired, it can also be achieved as shown in the completely connected mesh network of figure 8. In this network, the receive port of each module is connected to every other send port, and vice versa. The disadvantage of the complete connection is that a single bus and arbitration chain must encompass all modules. This implies that only one communication at a time can occur in the network. For increased communications concurrency a partially connected network would be more attractive such as that shown in figure 9. Such networks are truly multibus, with some unidirectional sections and some bidirectional.

Finally, as an important special case we draw attention to the common master slaves connection with bidirectional information transfer. As shown implemented with our single chip design in figure 10, this is achieved by using extra chips. In some cases, such as a master slave configuration with multiple slaves, it may be necessary for the slaves to identify themselves when sending to the master. Of course this could most easily be accomplished if the slaves had different programs in their ROMs including their own ID's. If the slaves had identical ROM contents however identification could still be achieved without an address bus as follows: At reset time an initializing routine in each slave could run a counter until the initial receipt of the arbitration token. It could then associate an address to itself based on the value of this count, and use this afterwards in the sender ID field of the packet headers (see figure 5).

#### IV. Pure Firmware Model

Here we examine whether the communication objectives set out above could be achieved without the extra specialized on-chip hardware. For this purpose we assume again the single chip computer with the two parallel bidirectional ports.

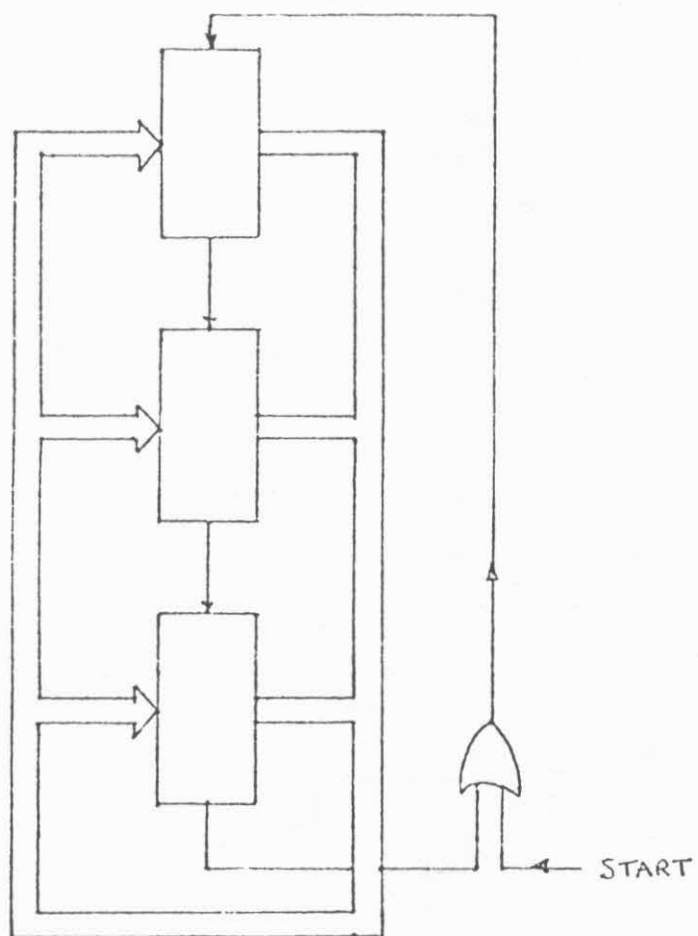
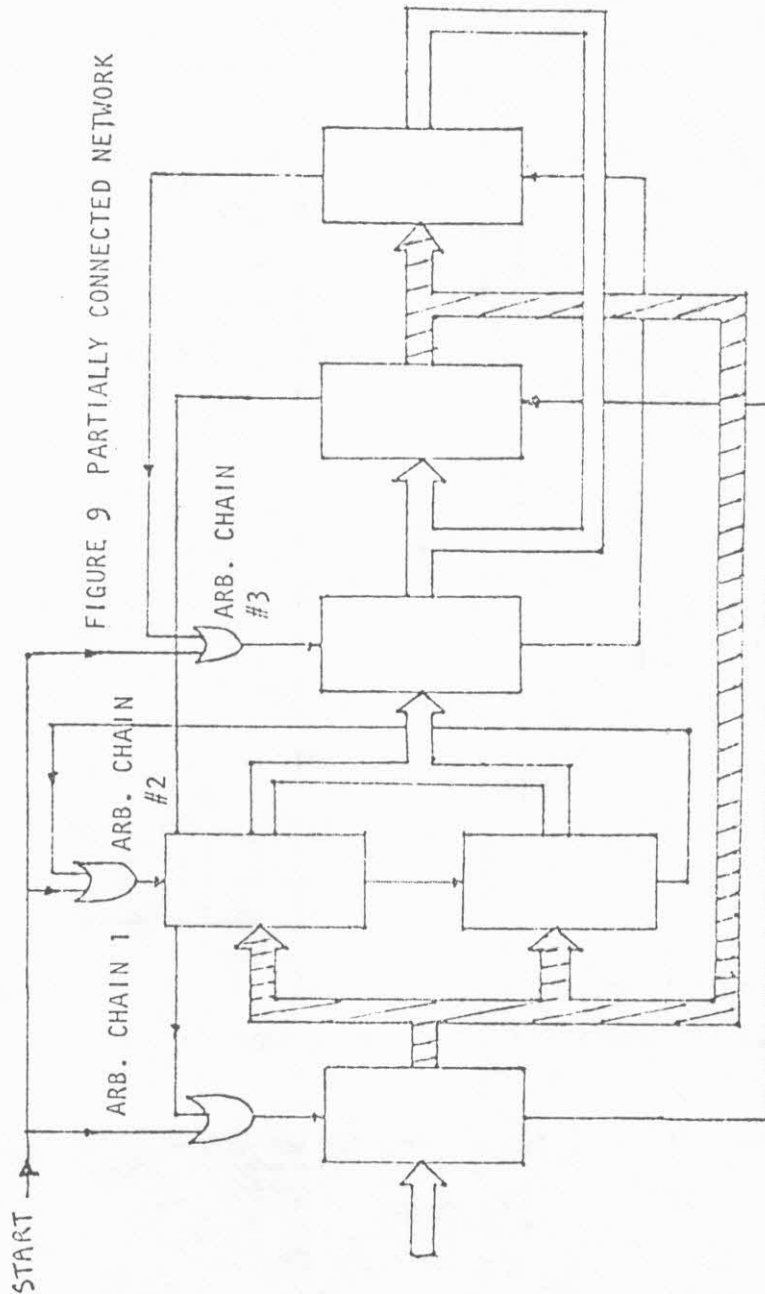


FIGURE 8 COMPLETELY CONNECTED NETWORK



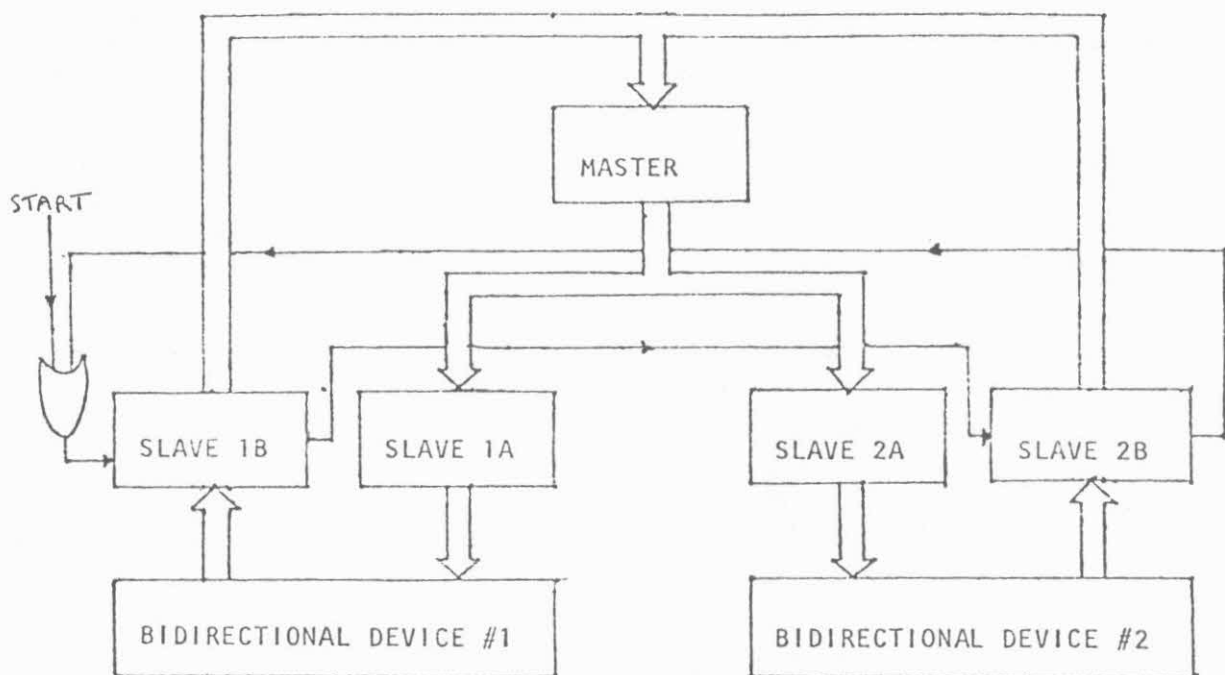


FIGURE 10 MASTER - SLAVE CONTROLLER

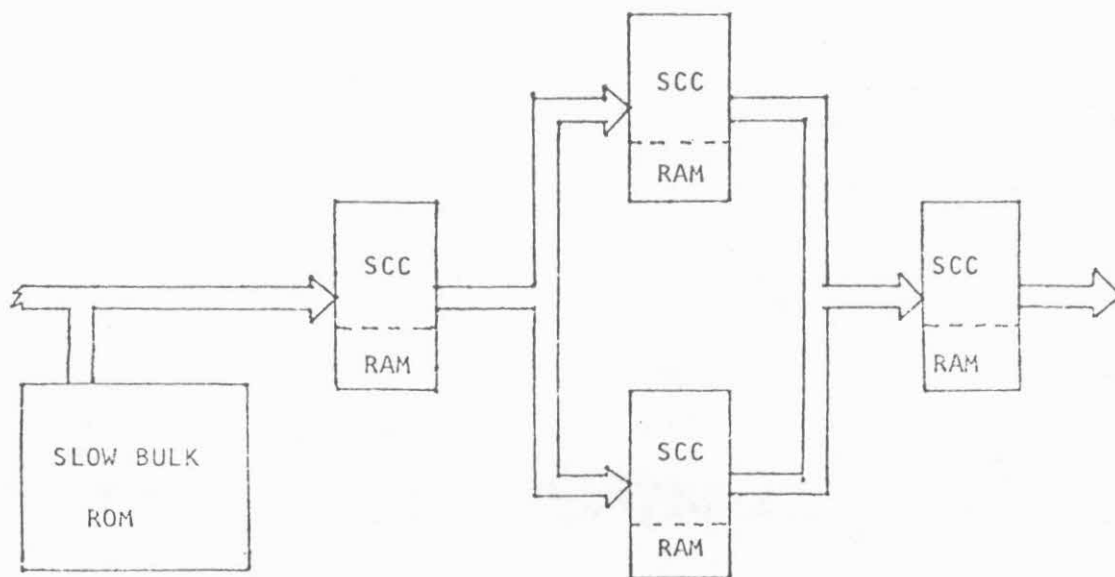


FIGURE 11 DOWNLOADING NETWORK

Without the dedicated hardware, communication protocols will necessarily be more complicated. As in other multicomputer environments we have to consider the questions of communication bus arbitration, module address selection, and the direction of information flow. Figure 12 shows our proposed utilization of the two ports to perform multicomputer communications. In the basic single chip computer, both ports A and B are bidirectional. For our present purpose port A would be the communication data bus and port B would be the control bus. Arbitration would be done in a fixed priority approach and only one-to-one module communication would be allowed.

Communications between modules would proceed in three phases. During the first phase, arbitration of the communication bus would be performed. After a new bus master is assigned, the second phase is entered and the master would select the module with which it wanted to communicate. In the third phase, communications between the master and the selected module would be performed word by word on a handshaking basis. All modules would participate in the first and second phase whereas only the master and the selected module will be involved in the third phase. Detailed description of the protocol can be found in appendix 2.

The size of the communications control program will be about 300 words. This estimation is based on the simple instructions used in section V to estimate the size of the arbitration program for the dedicated arbitrator. The timing of the arbitration process depends on the number of modules present. For example, if there were four modules on the bus, the time needed for the arbitration would be around 30 microseconds, assuming expected cycle times of next generation single chips. The address selection process would take at least another 3 microseconds. The timing of the message transmission would depend on the size of the message packet.

The advantage of this protocol is that no hardware modification is necessary. However, since only one communication bus can be available, no pipeline or synchro-parallel configurations can be achieved. Also the arbitration and address selection processes described above would have to be emulated by software and all modules would have to participate whether they were parties to that communication or not. This would result in a large amount of time used up in communications control. These must be accounted severe restrictions of the pure firmware case and an indication for hardware support of some kind.

#### V. Upgraded Chip with FIFO and Dedicated Arbitration.

In this section we will examine a third case in which it is assumed that the technology would permit the inclusion on the chip of a first-in first-out buffer store to assist in the communication of the message packet. Since arbitration could now take place at the packet rather than the word level, the timing requirements are slowed to the point where they could be handled by a software rather than a hardware process, - in fact on an identical chip, specialized by its ROM program for arbitration control.



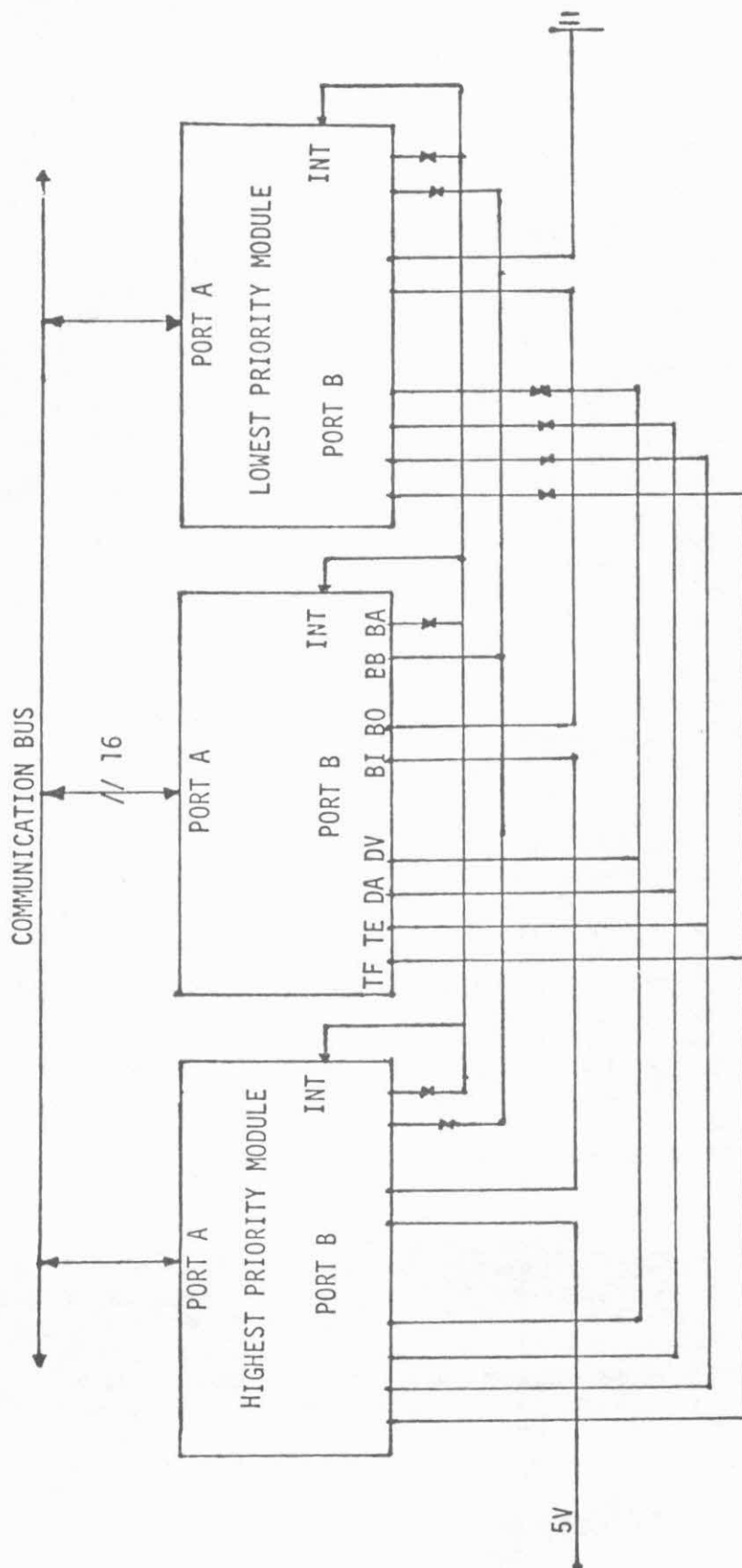


Figure 12. - PURE FIRMWARE MODEL

Otherwise we make similar assumptions as before, namely word parallel 16 bit communications and a 48 pin package limitation. Again one of the interrupt inputs will have to be multifunctional and during multicomputer mode serve as a TRANSMISSION-END signal input.

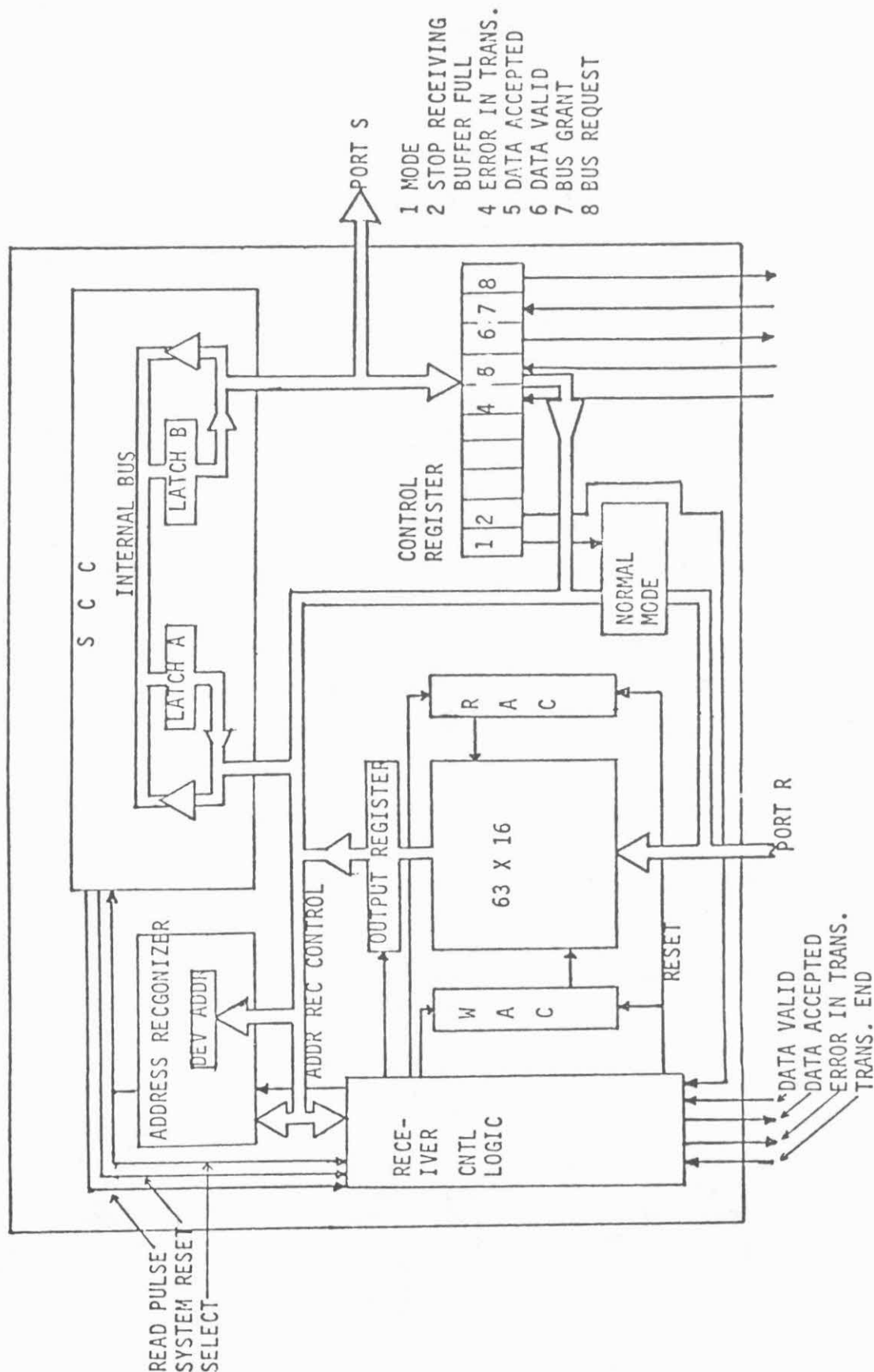
Figure 13 is the block diagram of the upgraded single chip microcomputer with FIFO. The module has two ports, namely port S for message transmitting and port R for message receiving. Both ports can be changed to bidirectional ports by software. The block labelled 'SCC' at the upper right corner is a dual port computer. To the left of SCC is the address recognizer hardware which is used to perform address comparison. If an address match occurs, it interrupts the SCC to input the message in the FIFO. If the device is not selected, the FIFO will be cleared for a new message packet. At the lower right corner is the receiver control hardware whose function is to control the address recognizer hardware and FIFO, and to generate receiver handshaking signals. In the lower center is the receiver FIFO and its control register. The structure of the FIFO is similar to current industry FIFO's. The input data is written into the storage array in a location specified by the Write Address Counter (WAC). The current output word is automatically available at the output register. After the current word of data is used, the next output word is read from the storage array at the location specified by the Read Address Counter (RAC). To the right of the FIFO hardware is the control register for the upgraded hardware. Part of the control register's contents is used as handshaking signals for transmitting messages.

Only one FIFO is placed on the receiver side in this scheme with the message words separately handshaken across under direct program control of the transmitter. This of course would be slower than if the message packet were transmitted between two FIFO's under direct hardware mediation. However we are using the FIFO here principally to simplify the control and we felt that two intermediary FIFO's would be more complex. The single FIFO scheme would still be faster than our first case because of the simpler synchronism -both of the communicating software processes do not have to attend at the same time. The receiver module can therefore poll the FIFO later and bring in the message with the fast repeat-I/O instruction. Thus in an N processor multibus system a bus concurrency close to N might be approached instead of being limited to N/2 as before. The detailed communications protocol for this case is given in Appendix 3.

#### Use of upgraded microcomputer as bus arbitrator

As we proposed in previous sections, our message communications scheme will be in packet format. There is a considerable amount of time lap between different packets being transmitted, which implies that the arbitration process can be achieved by software instead of dedicated hardware. In this section, we are going to examine the possibilities of using the same upgraded single chip computer as our communication bus arbitrator. Firstly, different arbitration schemes are discussed. Secondly, the size and speed of the arbitration program will be studied.

Figure 13 UPGRADED SINGLE CHIP MICROCOMPUTER WITH FIFO



For use as an arbitrator, our upgraded computer will be operated in normal mode, which means it will have two bidirectional ports: A and B. Port A will be used as bus request port and port B will be used as bus granted port. Figure 14 shows how local communication buses are connected to the arbitrators. More than one arbitrator can be connected together hierarchically to produce a larger arbitration system. Figure 15 shows a possible arbitration system.

Two different arbitration schemes could be implemented: Fixed Priority and Round-Robin systems. Pin 16 of the arbitrator bus request's port could be used to indicate which mode has been set, and pin 16 of the arbitrator bus grant's port used to indicate that the current bus master has released the bus. The fixed priority mode is entered when pin 16 of the arbitrator bus request's port is set to logic high. In this mode, pin 1 will have the highest priority and pin 15 will have the lowest priority. When the current bus master releases the bus by lowering its bus request line, arbitration phase is entered and the request with highest priority will be honored. Round-robin mode is entered when pin 16 of the arbitrator's receiver is set to logic low. In this mode, all pins will have the same priority and requests will be honored in a circular fashion. By combining the two arbitration schemes we could implement a hierarchy arbitration system as in figure 15. Modules having the same priority level are connected to the same arbitrator which implements the Round-robin scheme, where the arbitrators themselves are connected to a master arbitrator which implements the fixed priority scheme. With this hierarchical parallel structure, we could connect any number of modules together and still have minimal delay time. Both round-robin and fixed priority algorithms have the same structure. While a current bus master is using the bus, the bus arbitrator will continuously perform next bus arbitration. As soon as the current master has finished, a new bus master can be assigned. With this approach, delay time can be minimized.

Arbitration program outlines can be found in appendix 4. The sizes of both programs are about the same. Since we have to incorporate both programs into the program memory, their total size is less than 200 words. Worst case arbitration delay time is when there is no pre-arbitration done, in this case the arbitration delay time will be less than 30 microseconds. When there is pre-arbitration done, which is the average or best case, the timing will be less than 4 microseconds. With this arbitration delay time, we concluded that the size of the FIFO should be efficiently be 64 words, since the time the transmitter takes to transmit one packet should be greater than one arbitration period.

## VI. Conclusion and Discussion.

Of the three schemes examined we believe the pure firmware model has been shown to have inefficiencies which might cancel out the advantages of the multiprocessor operation in the type of applications envisaged. If technology and marketing considerations would permit the 48 pin package then the inclusion of upgraded communication hardware would result in a more efficient system, and if a FIFO could be included, still

LOCAL BUS CONNECTION SCHEME

Figure 14.

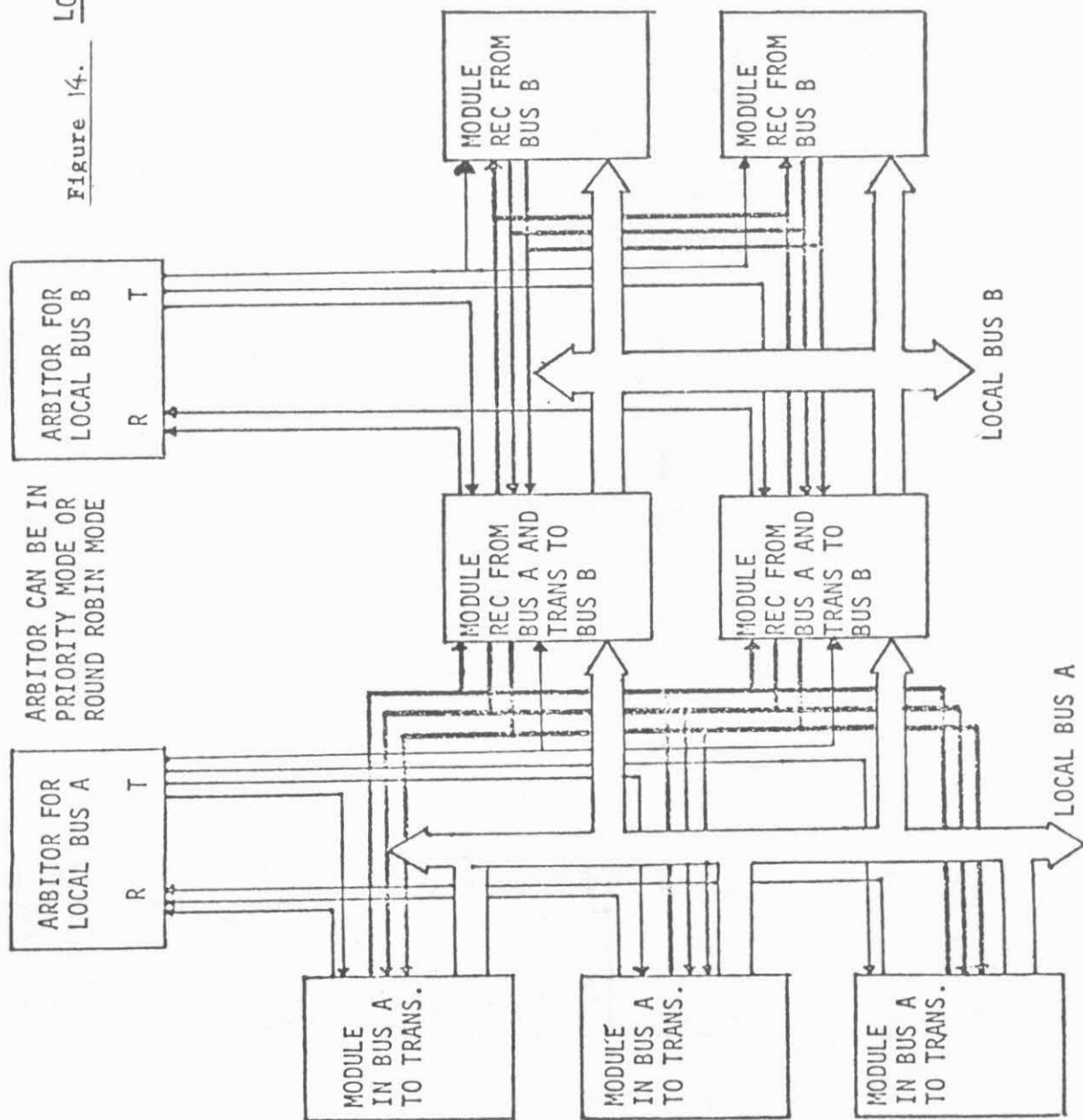
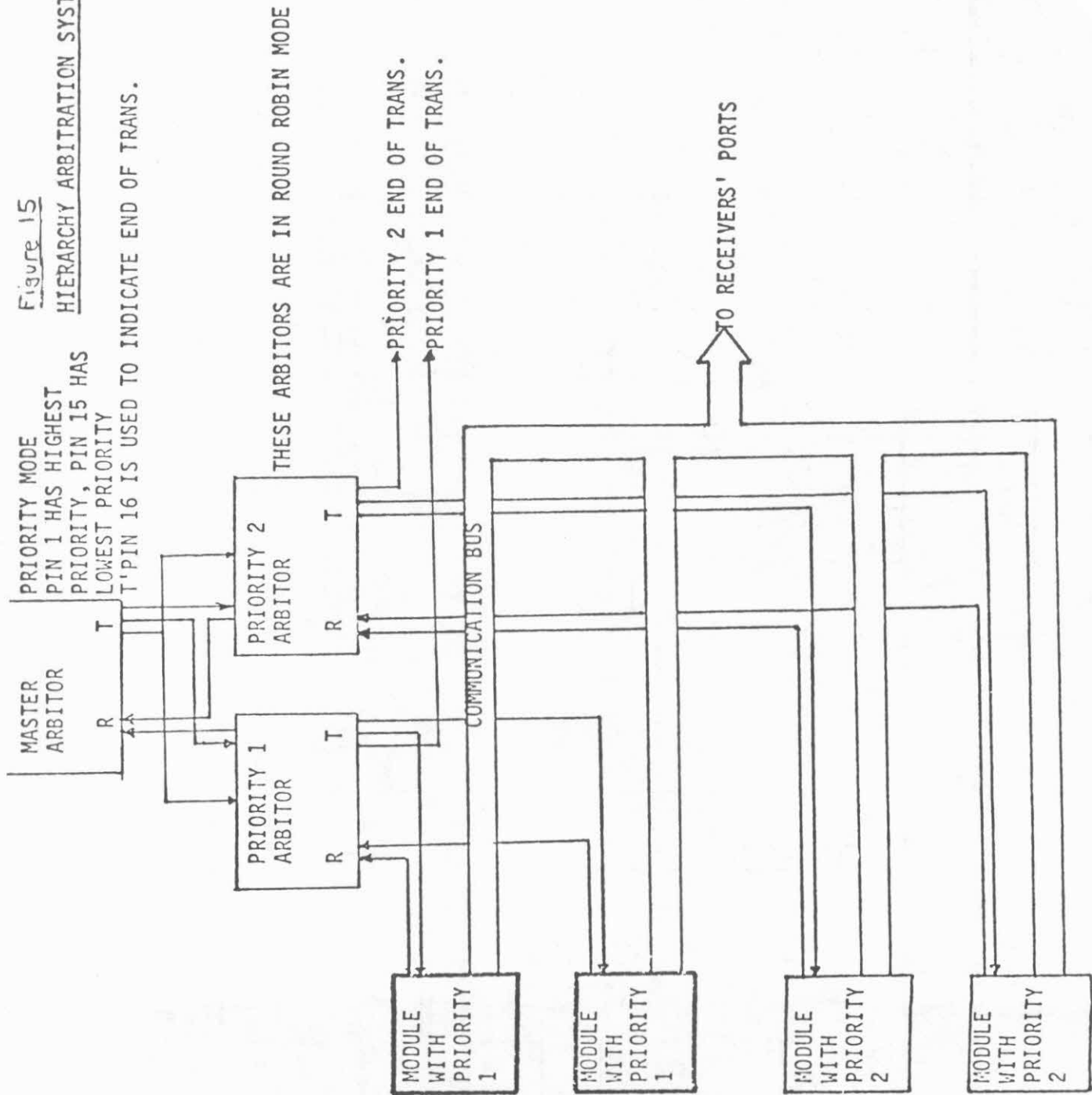


Figure 15

## HIERARCHY ARBITRATION SYSTEM

PRIORITY MODE  
PIN 1 HAS HIGHEST  
PRIORITY, PIN 15 HAS  
LOWEST PRIORITY  
T'PIN 16 IS USED TO



better. Realistically, these options may not be attractive in the first generation of 16 bit single chip computers since the communications software of two to three hundred words would be an appreciable fraction of the whole program memory space available. However, with each year that passes after that, with a doubling of program space, these schemes should become increasingly attractive.

In conclusion, what we have advocated here for the single chip computer can be viewed as a low cost alternative to the more expensive multiprocessor-multibus architectures which have been proposed for the current family of multichip microcomputers. It has also been oriented towards the area of fixed rather than general purpose applications. In this context, the development of a firmware kernel in ROM to remove most of the burdens of the communications from the users is clearly of crucial importance. Such a kernel would clearly have differences with kernels now being developed for large experimental general purpose multicomputers (such as [11]), since it should be oriented towards simplicity and speed efficiency.

An important property of the proposed chips is the flexibility of use and low cost. In this way the idea of employing extra chips of the same kind in a design can be easily entertained. Examples of this have been seen here for increasing communications fanout and for adding bidirectionality to the links. This perhaps may point the way to the use of such chips as this as general purpose system building blocks, much as discrete logic gates and programmed logic arrays have been used in previous technologies.

Looking slightly further ahead, we can see that one of the factors raising the costs of these multicomputer applications would be the different application software modules which would have to be burnt into chips in different parts of the network (parallel arms, serial arms). In other words we would like our standardized modules to be exactly alike in all respects. Eventually this might be achieved as shown in figure 11, using a system of downloading the applications software at power-on time into modules based on RAM rather than ROM. Here is where a hierarchical memory might be indicated in each chip with a large dense dynamic RAM backing up a small static RAM cache. This would also add new functions to the kernel, with perhaps a bootstrap version in each module bringing in the applications software and then the running kernel.

### References

- [1] Moore, G.E., "Are we ready for VLSI?", Proc. Caltech. Conference on Very Large Scale Integration, Jan. 1979, pp 3-14.
- [2] Mead C., & L. Conway, Introduction to VLSI systems, Addison-Wesley, 1980.
- [3] Clark J., "VLSI geometry processor for graphics", Computer J., July 1980, pp 59-68.
- [4] Caldwell J., "Real time text to speech using custom VLSI and standard micro-computers, proc. Spring Compcon 80, p43.
- [5] Dixon L.R., & T.B. Martin, Automatic speech and speaker recognition, IEEE press, 1979.
- [6] Medress M.F. et al, "A system for the recognition of spoken connected word sequences, ibid, pp 238-247.
- [7] Dixon L.R. & H.F. Siverman, "The 1976 acoustic processor MAP", IEEE Trans ASSP-25, (Oct 77), pp 367-379.
- [8] Roberts C.S., "An associative/parallel processor for partial match retrieval using superimposed codes", Proc. 7th Symp. Computer Architecture, (April 1980), pp 218-227.
- [9] Patterson D.A., & C.H. Sequin, "Design considerations for single chip computers of the future", IEEE Trans Vol. C-29:2 (Feb 1980), pp 108-116.
- [10] 8086 family user manual, Intel Corp, 1980.
- [11] Sadayappan P. et al, "An operating system kernel for a hierarchical computer", Proc. Fall Compcon 80, Sept 23-25 1980.
- [12] Sequin C.H., "message switching circuits for multimicrocomputers", Proc. Spring Compcon 80, pp 328-334.
- [13] Harris, J.A., & D.R. Smith, "Simulation experiments of a hierachical multicomputer" Proc. 6th Symp. Computer Architecture, 1979, pp .
- [14] Chen T.C., "Overlap and pipeline processing", Ch. 9 in Introduction to Computer Architecture, H.A. Stone, ed., SRA assoc., 2nd ed., 1980.



## APPENDIX 1

### Transmission:

\*\* The Kernel sets up the message block in its RAM. It will clear the READY-FOR-RECEIVING flag in the selection logic and set the WAIT-FOR-TOKEN flag in the daisy chain logic. It then continues processing and periodically polls the TOKEN-IN bit in the I/O register. When the daisy chain logic receives the token, it will check its WAIT-FOR-TOKEN flag. If the flag is set, it will set the TOKEN-IN bit in the I/O register and clear the WAIT-FOR-TOKEN flag.

\*\* When kernel knows the TOKEN-IN bit is set, it will change its transmitter port and handshaking lines from high impedance to logic false. RTL1 interrupt line is also disabled. Afterthis, kernel will load latch A with first word of message and set the DATA-VALID out bit to true. (First word of message block is the bit pattern specifying which subset of receiver is chosen)

\*\* The kernel will initiate a timer at this instance. If the timer has run out before the DATA-ACCEPTED IN bit is set, it will abort the message transfer and reset the appropriate bits and flags.

\*\* When the kernel sees its DATA-ACCEPTED IN bit is set, it will acknowledge by clearing its DATA-VALID OUT bit.

\*\* When the kernel see its DATA-ACCEPTED IN bit is cleared, it will send the rest of the message by handshaking. When transmitting the second word of the message, it will check the TRANSMISSION-ERROR IN bit. If the bit is set, it will clear the DATA-VALID OUT bit and check the last word transferred. As soon as the TRANSMISSION-ERROR IN bit is cleared by the receiver, the kernel will set the DATA-VALID OUT bit and re-transmit that word to the receiver. The kernel has control over number of retries and it will abort the message transfer if that limit has reached.

\*\* When the kernel is finished with the transmission, it will clear the TOKEN-IN bit and set the READY-FOR-RECEIVING flag in the selection logic. After this, it will put its transmitter port and handshaking lines back to high impedance state.

\*\* Upon the TOKEN-IN bit is cleared, the daisy chain logic will pass the token to the next module down the chain.

### Receiving:

\*\* The SELECT input line of the receiver will be in high impedance state during receiving and will be in logic false state when waiting for input.

\*\* When the receiver's SELECT input is on, it will check the READY-FOR-RECEIVING flag. If the flag is set, it will change the DATA-ACCEPTED OUT line from high impedance to logic true. In addition, the

receiver's port and handshaking lines are all changed to low impedance. If the READY-FOR-RECEIVING flag is not set, kernel will set the DATA-ACCEPTED OUT line from high impedance to logic false and DATA-VALID IN line to low impedance. When the DATA-VALID IN bit is reset, the kernel will reset the receiver's handshaking line back to high impedance state.

\*\* The kernel will receive the rest of the message by handshaking with the transmitter. It will check for transmission error in the second word of the message. If error did occur, it will set the TRANSMISSION-ERROR OUT bit to true and discard that word. As soon as the DATA-VALID IN bit is cleared, it will clear the TRANSMISSION-ERROR OUT bit and waits for re-transmission of that word. When predetermined number of retries is over, kernel will abort the message receiving and reset its bits and flags.

\*\* When the message receipt is completed, the kernel will put its receiver port and handshaking lines back to high impedance state. The SELECT input is reset to logic false.

## APPENDIX 2 - Pure Firmware Model.

- \* When the system first boots up, all modules' BB, BA, DV, DA, TE and TF lines are not asserted. Except for the highest priority module's BI line is connected to 5V and the lowest priority module's BO line is connected to ground, all other modules' BI and BO lines are not asserted.
- \* When a module wants to gain access to the bus, first it has to check the BB (Bus Busy) line. If it is asserted, it has to wait for the current master to finish. If BB is not asserted, the module will assert its BA line. Since the BA line is connected to all modules' T1 interrupt line (including itself), all modules will enter an interrupt service routine which then begin the arbitration process.
- \* The module which asserted the BA line will reset it.
- \* All modules will initiate a count down timer for maximum arbitration process duration. When the count times out and the BB line hasn't been asserted, arbitration process will be terminated and all modules will resume their previous processes.
- \* Meanwhile all modules sample their BI (Bus arbitration In) input. When this signal is asserted for a module which did not request the bus, then that module will assert its BO (Bus arbitration Out) and wait for BB assertion or timer run out to occur. If this module was not the one which requested the bus, it will assert its BB line.
- \* As soon as the module gains control to the bus, it becomes the current bus master. It then puts a device address word on to the data bus and asserts its DV (Data Valid) line. When its DA (Data Accepted) line is asserted by the selected module, the master will

reset its DV line and returns to its calling process to perform the message communication. If the DA line doesn't assert when the timer run out occurs, then the arbitration process will be terminated and all modules will resume their previous processes.

- \* When a module's BB line is asserted by a new master, it will wait for the device address word to determine whether it is being selected. If a module is selected, it will assert its DA line. When the master resets its DV line, the selected module will enter a communication routine. Other modules will terminate the arbitration process and resume their previous processes.
- \* The DA lines are wired OR together, so only one to one module communication is allowed.
- \* Communication will be performed on word by word handshaking basis. At any time if the receiver module asserts the TE (Trans. Error) line, master will try to retransmit the last word for a predefined number of times. If still fail, communication will be aborted.
- \* When the communication is finished, the master asserts the TF (Trans. Finished) line and waits for DA line assertion. When its DA is asserted, it will reset its TF and BB line and the communication is concluded.

### APPENDIX 3 - Upgraded Chip with FIFO

The modules all have distinct coded addresses. Modules will generate their addresses during the system boot up phase. When the system first boots up, all modules on a local bus will assert their BR (Bus Request) lines. Depending on the arbitration scheme used by the local arbitrator, individual modules will be granted the bus in succession and can generate their addresses based on the time elapsed.

Communications will proceed in two phases. During the first phase or bus arbitration phase, a module which wants to use the bus asserts its BR line and waits for its BG (Bus Grant) line to be asserted by the arbitrator. Once the BG line is asserted, the selected module will enter phase two which is the communication phase.

Since receiving is performed by hardware and buffered by a FIFO, we expect all the modules on the receiving side are ready to receive when the transmitter is ready to transmit the message packet. However, if the Receiving Buffer Full bit of the control register is set, the receiver module has to wait till it is clear before it can accept any more message. Handshaking signals are used throughout the communications, and the receivers' DA (Data Accepted) lines are wired AND together so as to ensure all the receivers are ready.

When the module gains access to the local communication bus, it will send

out the message packet with a packet header word containing the coded destination address to the receiver modules. Depending on the coded address, message packets can be directed to an individual module or broadcasts to all of them.

At any time, if the transmitter's Error-in-Transmission line is asserted by one of its receiver modules, the transmitter will retransmit the last word up to a predefined number of times. If the error condition persists the transmitter will abort the transmission.

When the transmission is finished, the transmitter module will reset its BR line. The arbitrator will then reset the corresponding BG line and asserts its Transmission End line to all the receiving modules.

Once the Transmission End line is asserted by the arbitrator, all the receiving modules will check the packet header word and determine whether the message is intended for them. If the message is intended for a particular module its control logic will interrupt the SCC in order to ship the message from the FIFO to the SCC's memory. If the message is not directed to the receiver, the receiver control logic will clear the FIFO for another new message packet.

When a transmitter module is not the current bus master, its transmitter's handshaking lines are in high impedance state.

#### APPENDIX 4 - Arbitration programs

Listing of the arbitration program:

```
/* initialization phase, determine which arbitration scheme to use */
```

```
F-begin:          Load save-req with 000...000
                  Load temp with request word
                  temp <- temp logical AND 100...000
                  temp = 0?
                  (false) jump to f-wait-loop
                  (true ) jump to r-wait loop
```

```
/* enter Fixed Priority mode, look for bus request */
```

```
f-wait-loop:      load req-wd with request word
                  req-wd <- req-wd logical AND 011...111
                  req-wd = 0?
                  (false) jump to f-arb-1
                  (true ) jump to f-wait-loop
```

```

/* determine who gets the bus */

f-arb-1:      load arb-end with 010...000
              load arb-curl with 100...000
f-arb-1loop:  left circular shift arb-curl 1 bit
              temp <- arb-curl logical AND req-wd
              temp = 0?
              (false) jump to f-grant-bus
              (true ) arb-end = arb-curl?
              (false) jump to f-arb-1loop
              (true ) jump to f-wait-loop

/* grant the bus to the highest priority requesting module */

f-grant-bus:  load bus grant word with arb-curl
              load bus granted with 1
              load save-req with 000...000

/* now start doing pre-arbitration while the bus is busy */

f-arb-2:      load req-wd-2 with request word
              req-wd-2 <- req-wd-2 XOR arb-curl
              req-wd-2 <- req-wd-2 logical AND 011...111
              req-wd-2 = 0?
              (false) jump to f-continuel
              (true ) jump to f-arb-2
f-continuel:  load arb-end with 100...000
              load arb-cur2 with 100...000

/* determine whether the bus is being used */

f-test:      bus-granted = 1?
              (false) jump to f-arb-2loop
              (true ) jump to f-poll

/* check to see whether the current master has finished if finished, and
there is an outstanding bus request, grant the bus. Otherwise continue
pre-arbitration process */

f-poll:      load temp with request word
              temp <- temp logical AND arb-curl
              temp = 0?
              (false) jump to f-arb-2loop
              (true ) load bus grant word with 100...000
              load bus grant word with 000...000
              load arb-curl with 000...000
              load bus-granted with 0
              save-req <> 0?
              (false) jump to f-arb-2loop
              (true ) load arb-curl with save-req
              jump to f-grant-bus

```

```

/* perform pre-arbitration */

f-arb-2loop:      left circular shift arb-cur2 1 bit
                  arb-end = arb-cur2?
                  (false) jump to f-continue2
                  (true ) jump to f-arb-2
f-continue2:      temp <- arb-cur2 logical AND req-wd-2
                  temp = 0?
                  (false) jump to f-set-or
                  (true ) jump to f-test

/* save the request from pre-arbitration process */

f-set-or:          load save-req with arb-cur2
                  bus-granted <> 1?
                  (false) jump to f-arb-2
                  (true ) jump to f-grant-bus

/* enter Round-Robin mode, look for bus request */

r-wait-loop:       load req-wd with request word
                  req-wd <- req-wd logical AND 011...111
                  req-wd = 0?
                  (false) jump to r-arb-1
                  (true ) jump to r-wait-loop

/* determine who should the bus be granted */

r-arb-1:           load arb-end with 010...000
                  load arb-curl with 100...000
r-arb-1loop:       left circular shift arb-curl 1 bit
                  temp <- arb-curl logical AND req-wd
                  temp = 0?
                  (false) jump to r-grant-bus
                  (true ) arb-end = arb-curl?
                  (false) jump to r-arb-1loop
                  (true ) jump to r-wait-loop

/* grant the bus to the highest priority requesting module */

r-grant-bus:        load bus grant word with arb-curl
                  load bus granted with 1
                  load save-req with 000...000

/* now start doing pre-arbitration while the bus is busy */

r-arb-2:           load req-wd-2 with request word
                  req-wd-2 <- req-wd-2 XOR arb-curl
                  req-wd-2 <- req-wd-2 logical AND 011...111
                  req-wd-2 = 0?
                  (false) jump to r-continuel
                  (true ) jump to r-arb-2

```

Communications for Next Generation Single-Chip Computers

```
/* determine where to start pre-arbitration */
```

```
r-continuel:      bus-granted = 1?
                  (false) jump to r-new
                  (true ) load arb-cur2 with arb-cur1
                  jump to r-poll
```

```
r-new:           load arb-end with 100...000
                  load arb-cur2 with 100...000
                  jump to r-arb-2loop
```

```
/* check to see whether the current master has finished if finished, and
there is an outstanding bus request, grant the bus. Otherwise continue
pre-arbitration process */
```

```
r-poll:          load temp with request word
                  temp <- temp logical AND arb-cur1
                  temp = 0?
                  (false) jump to r-arb-2loop
                  (true ) load bus grant word with 100...000
                  load bus grant word with 000...000
                  load arb-cur1 with 000...000
                  load bus-granted with 0
                  save-req <> 0?
                  (false) jump to r-arb-2loop
                  (true ) load arb-cur1 with save-req
                  jump to r-grant-bus
```

```
/* perform pre-arbitration */
```

```
r-arb-2loop:      left circular shift arb-cur2 1 bit
                  arb-end = arb-cur2?
                  (false) jump to r-continue2
                  (true ) jump to r-arb-2
r-continue2:      temp <- arb-cur2 logical AND req-wd-2
                  temp = 0?
                  (false) jump to r-set-or
                  (true ) jump to r-test
```

```
/* save the request from pre-arbitration process */
```

```
r-set-or:         load save-req with arb-cur2
                  bus-granted <> 1?
                  (false) jump to r-arb-2
                  (true ) jump to r-grant-bus
```

Note: save-req, temp, req-wd, arb-end, arb-cur1, bus-granted, req-wd-2, arb-cur2 can be registers or memory words.





CONFERENCE PROCEEDINGS ORDER FORM

Send orders to:

Computer Science Librarian  
Caltech 256-80  
Pasadena CA 91125

Quantity

\_\_\_\_\_ 1979 Proceedings at \$25 (\$35 out of U.S.) \$ \_\_\_\_\_

\_\_\_\_\_ 1981 Proceedings at \$20 (\$30 out of U.S.) \$ \_\_\_\_\_

Check enclosed: Total \$ \_\_\_\_\_

Prices are postpaid. Please do not send purchase orders.

Please send to (type or print):

Name: \_\_\_\_\_

Address \_\_\_\_\_

\_\_\_\_\_

CONFERENCE PROCEEDINGS ORDER FORM

Send orders to:

Computer Science Librarian  
Caltech 256-80  
Pasadena CA 91125

Quantity

\_\_\_\_\_ 1979 Proceedings at \$25 (\$35 out of U.S.) \$ \_\_\_\_\_

\_\_\_\_\_ 1981 Proceedings at \$20 (\$30 out of U.S.) \$ \_\_\_\_\_

Check enclosed: Total \$ \_\_\_\_\_

Prices are postpaid. Please do not send purchase orders.

Please send to (type or print):

Name: \_\_\_\_\_

Address \_\_\_\_\_

\_\_\_\_\_

CONFERENCE PROCEEDINGS ORDER FORM

Send orders to:

Computer Science Librarian  
Caltech 256-80  
Pasadena CA 91125

Quantity

\_\_\_\_\_ 1979 Proceedings at \$25 (\$35 out of U.S.) \$ \_\_\_\_\_

\_\_\_\_\_ 1981 Proceedings at \$20 (\$30 out of U.S.) \$ \_\_\_\_\_

Check enclosed: Total \$ \_\_\_\_\_

Prices are postpaid. Please do not send purchase orders.

Please send to (type or print):

Name: \_\_\_\_\_

Address \_\_\_\_\_

\_\_\_\_\_

-----  
CONFERENCE PROCEEDINGS ORDER FORM

Send orders to:

Computer Science Librarian  
Caltech 256-80  
Pasadena CA 91125

Quantity

\_\_\_\_\_ 1979 Proceedings at \$25 (\$35 out of U.S.) \$ \_\_\_\_\_

\_\_\_\_\_ 1981 Proceedings at \$20 (\$30 out of U.S.) \$ \_\_\_\_\_

Check enclosed: Total \$ \_\_\_\_\_

Prices are postpaid. Please do not send purchase orders.

Please send to (type or print):

Name: \_\_\_\_\_

Address \_\_\_\_\_

\_\_\_\_\_



CONFERENCE PROCEEDINGS ORDER FORM

Send orders to:

Computer Science Librarian  
Caltech 256-80  
Pasadena CA 91125

Quantity

\_\_\_\_\_ 1979 Proceedings at \$25 (\$35 out of U.S.) \$ \_\_\_\_\_

\_\_\_\_\_ 1981 Proceedings at \$20 (\$30 out of U.S.) \$ \_\_\_\_\_

Check enclosed: Total \$ \_\_\_\_\_

Prices are postpaid. Please do not send purchase orders.

Please send to (type or print):

Name: \_\_\_\_\_

Address \_\_\_\_\_

\_\_\_\_\_

-----  
CONFERENCE PROCEEDINGS ORDER FORM

Send orders to:

Computer Science Librarian  
Caltech 256-80  
Pasadena CA 91125

Quantity

\_\_\_\_\_ 1979 Proceedings at \$25 (\$35 out of U.S.) \$ \_\_\_\_\_

\_\_\_\_\_ 1981 Proceedings at \$20 (\$30 out of U.S.) \$ \_\_\_\_\_

Check enclosed: Total \$ \_\_\_\_\_

Prices are postpaid. Please do not send purchase orders.

Please send to (type or print):

Name: \_\_\_\_\_

Address \_\_\_\_\_

\_\_\_\_\_

